Contents

Welcome to Puppet Enterprise® 2019.8.12 ................................................................. 7
  PE software architecture .............................................................................................. 7
  Component versions in recent PE releases .............................................................. 14
  FIPS 140-2 enabled PE ............................................................................................... 17
  Cloud PE ...................................................................................................................... 17
  Getting support .......................................................................................................... 20
  Using the PE docs ....................................................................................................... 28
  Puppet platform documentation for PE ...................................................................... 30
  API index ...................................................................................................................... 32

Release notes .............................................................................................................. 35
  PE release notes .......................................................................................................... 35
  PE known issues .......................................................................................................... 63
  What's new since PE 2018.1 ......................................................................................... 67
  Platform release notes ............................................................................................... 73
    Puppet release notes ................................................................................................. 74
    Puppet known issues .................................................................................................. 137
    Puppet Server release notes ..................................................................................... 138
    Puppet Server known issues ..................................................................................... 148
    Facter release notes .................................................................................................. 149
    Facter known issues .................................................................................................. 156

Getting started with Puppet Enterprise ....................................................................... 157
  Install PE .................................................................................................................... 158
  Add nodes to the inventory ......................................................................................... 160
  Add code and set up Code Manager ......................................................................... 161
  Manage Apache configuration on *nix targets ......................................................... 164
    Install the apache module ......................................................................................... 165
    Set up Apache node groups ....................................................................................... 166
    Organize webserver configurations with roles and profiles ..................................... 167
  Manage IIS configuration on Windows targets ......................................................... 170
    Install the iis module ................................................................................................. 171
    Set up IIS node groups ............................................................................................... 172
    Organize webserver configurations with roles and profiles ..................................... 173
  Next steps ................................................................................................................... 177

Installing ....................................................................................................................... 177
  Supported architectures .............................................................................................. 178
  System requirements ................................................................................................. 182
    Hardware requirements ............................................................................................... 182
    Supported operating systems ..................................................................................... 184
    Supported browsers .................................................................................................... 190
    System configuration .................................................................................................. 190
  What gets installed and where? ................................................................................. 202
  Installing PE ................................................................................................................. 209
  Installing PE in cloud environments ........................................................................... 219
Purchasing and installing a license key........................................................................................................ 222
Installing agents....................................................................................................................................... 223
Installing compilers............................................................................................................................. 244
Installing client tools.......................................................................................................................... 250
Uninstalling........................................................................................................................................... 254

Upgrading................................................................................................................................................. 256
Upgrading Puppet Enterprise.................................................................................................................. 257
Upgrading agents.................................................................................................................................. 273

Configuring Puppet Enterprise........................................................................................................ 279
Tuning infrastructure nodes.................................................................................................................. 279
Methods for configuring PE................................................................................................................ 286
Configuring Puppet Server................................................................................................................... 288
Configuring PuppetDB.......................................................................................................................... 292
Configuring security settings.............................................................................................................. 294
Configuring proxies............................................................................................................................. 298
Configuring the console......................................................................................................................... 299
Configuring orchestration.................................................................................................................... 302
Configuring ulimit.................................................................................................................................. 306
Writing configuration files.................................................................................................................... 307
Analytics data collection...................................................................................................................... 308
Static catalogs....................................................................................................................................... 312

Configuring disaster recovery........................................................................................................ 315
Disaster recovery................................................................................................................................. 315
Configure disaster recovery............................................................................................................... 324

Accessing the console.......................................................................................................................... 329
Reaching the console............................................................................................................................ 329
Logging in.............................................................................................................................................. 330

Managing access...................................................................................................................................... 331
User permissions and user roles........................................................................................................... 331
Creating and managing local users and user roles............................................................................. 338
Connecting external directory services to PE...................................................................................... 340
Working with user groups from an external directory service........................................................... 346
Token-based authentication................................................................................................................... 347
RBAC API............................................................................................................................................... 353
Forming RBAC API requests.................................................................................................................. 354
RBAC service errors............................................................................................................................. 355
RBAC API v1......................................................................................................................................... 358
RBAC API v2......................................................................................................................................... 378
Activity service API............................................................................................................................. 382
Forming activity service API requests.................................................................................................. 382
Event types reported by the activity service......................................................................................... 383
Events endpoints................................................................................................................................. 386

Monitoring and reporting.................................................................................................................... 395
Monitoring infrastructure state............................................................................................................. 396
Managing nodes .................................................................................................... 451

Adding and removing agent nodes ..................................................................... 452
Adding and removing agentless nodes ................................................................. 454
How nodes are counted ........................................................................................ 457
Running Puppet on nodes ..................................................................................... 458
Grouping and classifying nodes ............................................................................ 460
Making changes to node groups .......................................................................... 468
Environment-based testing .................................................................................... 470
Preconfigured node groups .................................................................................. 472
Managing Windows nodes ................................................................................... 477
Designing system configs: roles and profiles ......................................................... 502
The roles and profiles method .............................................................................. 502
Roles and profiles example .................................................................................... 505
Designing advanced profiles ............................................................................... 508
Designing convenient roles .................................................................................. 524
Node classifier API v1 ............................................................................................ 528
Forming node classifier requests .......................................................................... 529
Groups endpoint ................................................................................................... 530
Groups endpoint examples ................................................................................... 548
Classes endpoint ................................................................................................... 550
Classification endpoint ........................................................................................ 552
Commands endpoint ............................................................................................. 563
Environments endpoint ......................................................................................... 564
Nodes check-in history endpoint .......................................................................... 565
Group children endpoint ....................................................................................... 567
Rules endpoint ....................................................................................................... 571
Import hierarchy endpoint .................................................................................... 572
Last class update endpoint .................................................................................... 572
Update classes endpoint ....................................................................................... 573
Validation endpoints ............................................................................................. 574
Node classifier errors ........................................................................................... 577
Node classifier API v2 ............................................................................................ 578
Classification endpoint ........................................................................................ 579
Node inventory API ............................................................................................... 581

Managing patches ............................................................................................... 592

Configuring patch management .......................................................................... 593
Patching nodes ..................................................................................................... 599
### Orchestrating Puppet runs, tasks, and plans

- How Puppet orchestrator works ............................................................... 604
- Setting up the orchestrator workflow .................................................. 607
- Configuring Puppet orchestrator ............................................................ 614
- Running Puppet on demand ................................................................. 620
  - Running Puppet on demand from the console ........................................ 621
  - Running Puppet on demand from the CLI .............................................. 627
  - Running Puppet on demand with the API ............................................ 630
- Tasks in PE .............................................................................................. 633
  - Installing tasks ..................................................................................... 634
  - Running tasks in PE ............................................................................. 634
  - Writing tasks ....................................................................................... 645
- Plans in PE ............................................................................................... 663
  - Plans in PE versus Bolt plans ............................................................. 664
  - Installing plans .................................................................................... 666
  - Running plans in PE ........................................................................... 667
  - Writing plans ...................................................................................... 670
- Orchestrator API v1 endpoints ............................................................... 695
  - Forming orchestrator API requests ..................................................... 696
  - Command endpoints ........................................................................... 697
  - Events endpoints ................................................................................ 715
  - Inventory endpoints ............................................................................ 721
  - Jobs endpoints .................................................................................... 723
  - Scheduled jobs endpoints .................................................................. 735
  - Plans endpoints ................................................................................... 740
  - Plan jobs endpoints ............................................................................ 743
  - Tasks endpoints .................................................................................. 750
  - Root endpoints ................................................................................... 754
  - Usage endpoints .................................................................................. 755
  - Scopes endpoints ............................................................................... 757
  - Orchestrator API error responses ....................................................... 760
- Migrating Bolt tasks and plans to PE ...................................................... 760

### Managing and deploying Puppet code

- Managing environments with a control repository .................................. 764
- Managing environment content with a Puppetfile ................................... 768
- Managing code with Code Manager ....................................................... 774
  - How Code Manager works .................................................................. 774
  - Set up Code Manager .......................................................................... 777
  - Configure Code Manager ..................................................................... 777
  - Lockless code deploys ......................................................................... 781
  - Customize Code Manager configuration in Hiera .................................. 784
  - Triggering Code Manager on the command line .................................... 792
  - Triggering Code Manager with a webhook ......................................... 798
  - Triggering Code Manager with custom scripts .................................... 799
  - Troubleshooting Code Manager ......................................................... 801
  - Code Manager API .............................................................................. 804
- Managing code with r10k ...................................................................... 816
  - Set up r10k ......................................................................................... 817
  - Configure r10k ................................................................................... 817
  - Customizing r10k configuration ......................................................... 818
  - Deploying environments with r10k .................................................... 825
  - r10k command reference .................................................................... 827
SSL and certificates

- Regenerate the console certificate
- Regenerate infrastructure certificates
- Use an independent intermediate certificate authority
- Use a custom SSL certificate for the console
- Generate a custom Diffie-Hellman parameter file
- Enable TLSv1

Maintenance

- Backing up and restoring PE
- Database maintenance

Troubleshooting

- Log locations
- Troubleshooting installation
- Troubleshooting disaster recovery
- Troubleshooting puppet infrastructure run commands
- Troubleshooting connections between components
- Troubleshooting the databases
- Troubleshooting cloud deployments
- Troubleshooting Windows
- Troubleshooting backup and restore
Welcome to Puppet Enterprise® 2019.8.12

Puppet Enterprise (PE) helps you be productive, agile, and collaborative while managing your IT infrastructure. PE combines a model-driven approach with imperative task execution so you can effectively manage hybrid infrastructure across its entire lifecycle. PE provides the common language that all teams in an IT organization can use to successfully adopt practices such as version control, code review, automated testing, continuous integration, and automated deployment.

<table>
<thead>
<tr>
<th>Puppet Enterprise docs links</th>
<th>Other useful links</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Getting started</strong></td>
<td>Docs for related Puppet products</td>
</tr>
<tr>
<td>Release notes</td>
<td>Open source Puppet</td>
</tr>
<tr>
<td>Architecture overview</td>
<td>Continuous Delivery for Puppet Enterprise</td>
</tr>
<tr>
<td>System requirements</td>
<td>Puppet Remediate</td>
</tr>
<tr>
<td>Getting started guide</td>
<td>Bolt</td>
</tr>
<tr>
<td><strong>Install and configure PE</strong></td>
<td>Puppet Development Kit</td>
</tr>
<tr>
<td>Install PE</td>
<td></td>
</tr>
<tr>
<td>Install agents</td>
<td></td>
</tr>
<tr>
<td>Add agentless nodes</td>
<td></td>
</tr>
<tr>
<td>Configure and tune PE</td>
<td></td>
</tr>
<tr>
<td><strong>Manage your infrastructure</strong></td>
<td></td>
</tr>
<tr>
<td>Manage nodes</td>
<td>Share and contribute</td>
</tr>
<tr>
<td>Run jobs, tasks, and plans</td>
<td></td>
</tr>
<tr>
<td>Deploy Puppet code</td>
<td></td>
</tr>
</tbody>
</table>

PE software architecture

Puppet Enterprise (PE) is made up of various components and services including the primary server and compilers, the Puppet agent, console services, Code Manager and r10k, orchestration services, and databases.

The following diagram shows the architecture of a typical PE installation.
Related information

Component versions in recent PE releases on page 14
These tables show which components are in Puppet Enterprise (PE) 2019.y and 2018.y. releases. Component version tables for prior releases are available in the Documentation for other PE versions on page 30. Component version tables for the current STS and LTS releases are available in the latest PE documentation.

The primary server and compilers
The primary server is the central hub of activity and process in Puppet Enterprise. This is where code is compiled to create agent catalogs, and where SSL certificates are verified and signed.

PE infrastructure components are installed on a single node: the primary server. The primary server always contains a compiler and a Puppet Server. As your installation grows, you can add additional compilers to distribute the catalog compilation workload.

Each compiler contains the Puppet Server, the catalog compiler, and an instance of file sync.

Puppet Server
Puppet Server is an application that runs on the Java Virtual Machine (JVM) on the primary server. In addition to hosting endpoints for the certificate authority service, it also powers the catalog compiler, which compiles configuration catalogs for agent nodes, using Puppet code and various other data sources.

Catalog compiler
To configure a managed node, the agent uses a document called a catalog, which it downloads from the primary server or a compiler. The catalog describes the desired state for each resource that should be managed on the node, and it can specify dependency information for resources that should be managed in a certain order.

File sync
File sync keeps your code synchronized across multiple compilers. When triggered by a web endpoint, file sync takes changes from the working directory on the primary server and deploys the code to a live code directory. File sync then deploys that code to any compilers so that your code is deployed only when it's ready.
Certificate Authority

The internal certificate authority (CA) service accepts certificate signing requests (CSRs) from nodes, serves certificates and a certificate revocation list (CRL) to nodes, and optionally accepts commands to sign or revoke certificates.

The CA service uses .pem files in the standard ssldir to store credentials. You can use the puppetserver ca command to interact with these credentials, including listing, signing, and revoking certificates.

Note: Depending on your architecture and security needs, the CA can be hosted either on the primary server or on its own node. The CA service on compilers is configured, by default, to proxy CA requests to the CA.

Related information

Hardware requirements on page 182
These hardware requirements are based on internal testing at Puppet and are provided as minimum guidelines to help you determine your hardware needs.

Installing compilers on page 244
As your Puppet Enterprise infrastructure scales up to 4,000 nodes and beyond, add load-balanced compilers to your installation to increase the number of agents you can manage.

About file sync on page 828
File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

The Puppet agent

Managed nodes run the Puppet agent application, usually as a background service. The primary server and any compilers also run a Puppet agent.

Periodically, the agent sends facts to a primary server and requests a catalog. The primary server compiles the catalog using several sources of information, and returns the catalog to the agent.

After it receives a catalog, the agent applies it by checking each resource the catalog describes. If it finds any resources that are not in their desired state, it makes the changes necessary to correct them. (Or, in no-op mode, it reports on what changes would have been made.)

After applying the catalog, the agent submits a report to its primary server. Reports from all the agents are stored in PuppetDB and can be accessed in the console.

Puppet agent runs on *nix and Windows systems.

• Puppet Agent on *nix Systems
• Puppet Agent on Windows Systems

Facter

Facter is the cross-platform system profiling library in Puppet. It discovers and reports per-node facts, which are available in your Puppet manifests as variables.

Before requesting a catalog, the agent uses Facter to collect system information about the machine it’s running on.

For example, the fact os returns information about the host operating system, and networking returns the networking information for the system. Each fact has various elements to further refine the information being gathered. In the networking fact, networking.hostname provides the hostname of the system.

Facter ships with a built-in list of core facts, but you can build your own custom facts if necessary.

You can also use facts to determine the operational state of your nodes and even to group and classify them in the NC.
Console services

The console services includes the console, role-based access control (RBAC) and activity services, and the node classifier.

The console

The console is the web-based user interface for managing your systems.

The console can:

• browse and compare resources on your nodes in real time.
• analyze events and reports to help you visualize your infrastructure over time.
• browse inventory data and backed-up file contents from your nodes.
• group and classify nodes, and control the Puppet classes they receive in their catalogs.
• manage user access, including integration with external user directories.

The console leverages data created and collected by PE to provide insight into your infrastructure.

RBAC

In PE, you can use RBAC to manage user permissions. Permissions define what actions users can perform on designated objects.

For example:

• Can the user grant password reset tokens to other users who have forgotten their passwords?
• Can the user edit a local user's role or permissions?
• Can the user edit class parameters in a node group?

The RBAC service can connect to external LDAP directories. This means that you can create and manage users locally in PE, import users and groups from an existing directory, or do a combination of both. PE supports OpenLDAP and Active Directory.

You can interact with the RBAC and activity services through the console. Alternatively, you can use the RBAC service API and the activity service API. The activity service logs events for user roles, users, and user groups.

PE users generate tokens to authenticate their access to certain command line tools and API endpoints. Authentication tokens are used to manage access to the following PE services and tools: Puppet orchestrator, Code Manager, Node Classifier, role-based access control (RBAC), and the activity service.

Authentication tokens are tied to the permissions granted to the user through RBAC, and provide users with the appropriate access to HTTP requests.

Node classifier

PE comes with its own node classifier (NC), which is built into the console.

Classification is when you configure your managed nodes by assigning classes to them. Classes provide the Puppet code—distributed in modules—that enable you to define the function of a managed node, or apply specific settings and values to it. For example, you might want all of your managed nodes to have time synchronized across them. In this case, you would group the nodes in the NC, apply an NTP class to the group, and set a parameter on that class to point at a specific NTP server.

You can create your own classes, or you can take advantage of the many classes that have already been created by the Puppet community. Reduce the potential for new bugs and to save yourself some time by using existing classes from modules on the Forge, many of which are approved or supported by Puppet by Perforce.

You can also classify nodes using the NC API.

Related information

Managing access on page 331
Role-based access control, more succinctly called RBAC, is used to grant individual users the permission to perform specific actions. Permissions are grouped into user roles, and each user is assigned at least one user role.

**Managing nodes** on page 451

Common node management tasks include adding and removing nodes from your deployment, grouping and classifying nodes, and running Puppet on nodes. You can also deploy code to nodes using an environment-based testing workflow or the roles and profiles method.

**Code Manager and r10k**

PE includes tools for managing and deploying your Puppet code: Code Manager and r10k.

These tools install modules, create and maintain environments, and deploy code to your primary servers, all based on code you keep in Git. They sync the code to your primary servers, so that all your servers start running the new code at the same time, without interrupting agent runs.

Both Code Manager and r10k are built into PE, so you don't have to install anything, but you need to have a basic familiarity with Git.

Code Manager comes with a command line tool which you can use to trigger code deployments from the command line.

**Related information**

**Managing and deploying Puppet code** on page 763

Puppet Enterprise (PE) includes built-in tools for managing and deploying your Puppet code. Code Manager and r10k are code management tools that automatically install modules, create and maintain environments, and deploy new code to your primary server and compilers, all based on version control of your Puppet code and data.

**Triggering Code Manager on the command line** on page 792

The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.

**How Puppet orchestrator works** on page 604

With the Puppet orchestrator, you can run Puppet, tasks, or plans on-demand.

**Orchestration services**

Orchestration services is the underlying toolset that manages Puppet runs, tasks, and plans, allowing you to make on-demand changes in your infrastructure.

For example, you can use it to enforce change on the environment level without waiting for nodes to check in for regular 30-min intervals, or use it to schedule a task on target nodes once per day.

The orchestration service interacts with PuppetDB to retrieve facts about nodes. To run orchestrator jobs, users must first authenticate to Puppet Access, which verifies their user and permission profile as managed in RBAC.

**Agentless Catalog Executor (ACE) service**

The ACE service enables you to run Puppet jobs, like tasks and plans, on nodes that don't have a Puppet agent installed on them. ACE service primarily runs through the orchestrator but it can be configured by itself. See the PE ACE server configuration docs to configure ACE.

**PE databases**

PE uses PostgreSQL as a database backend. You can use an existing instance, or PE can install and manage a new one.

The PE PostgreSQL instance includes the following databases:

<table>
<thead>
<tr>
<th>Database</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-activity</td>
<td>Activity data from the Classifier, including who, what and when</td>
</tr>
<tr>
<td>Database</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>pe-classifier</td>
<td>Classification data, all node group information</td>
</tr>
<tr>
<td>pe-puppetdb</td>
<td>Exported resources, catalogs, facts, and reports (see more, below)</td>
</tr>
<tr>
<td>pe-rbac</td>
<td>Users, permissions, and AD/LDAP info</td>
</tr>
<tr>
<td>pe-orchestrator</td>
<td>Details about job runs, users, nodes, and run results</td>
</tr>
</tbody>
</table>

**PuppetDB**

PuppetDB collects data generated throughout your Puppet infrastructure. It enables advanced features like exported resources, and is the database from which the various components and services in PE access data. Agent run reports are stored in PuppetDB.

See the PuppetDB overview for more information.

**Related information**

[Database maintenance](#) on page 845

You can optimize the Puppet Enterprise (PE) databases to improve performance.

**Security and communications**

The services and components in PE use a variety of communication and security protocols.

<table>
<thead>
<tr>
<th>Service/Component</th>
<th>Communication Protocol</th>
<th>Authentication</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet Server</td>
<td>HTTPS, SSL/TLS</td>
<td>SSL certificate verification with Puppet CA</td>
<td>trapperkeeper-auth</td>
</tr>
<tr>
<td>Certificate Authority</td>
<td>HTTPS, SSL/TLS</td>
<td>SSL certificate verification with Puppet CA</td>
<td>trapperkeeper-auth</td>
</tr>
<tr>
<td>Puppet agent</td>
<td>SSL/TLS</td>
<td>SSL certificate verification with Puppet CA</td>
<td>n/a</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>HTTPS externally, or HTTP on the loopback interface</td>
<td>SSL certificate verification with Puppet CA</td>
<td>SSL certificate whitelist</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>PostgreSQL TCP, SSL for PE</td>
<td>SSL certificate verification with Puppet CA</td>
<td>SSL certificate whitelist</td>
</tr>
<tr>
<td>Activity service</td>
<td>SSL</td>
<td>SSL certificate verification with Puppet CA, token authentication</td>
<td>RBAC user-based authorization</td>
</tr>
<tr>
<td>RBAC</td>
<td>SSL</td>
<td>SSL certificate verification with Puppet CA, token authentication</td>
<td>RBAC user-based authorization</td>
</tr>
<tr>
<td>Classifier</td>
<td>SSL</td>
<td>SSL certificate verification with Puppet CA, token authentication</td>
<td>RBAC user-based authorization</td>
</tr>
<tr>
<td>Console Services UI</td>
<td>SSL</td>
<td>Session-based authentication</td>
<td>RBAC user-based authorization</td>
</tr>
<tr>
<td>Orchestrateor</td>
<td>HTTPS, Secure web sockets</td>
<td>RBAC token authentication</td>
<td>RBAC user-based authorization</td>
</tr>
<tr>
<td>PXP agent</td>
<td>Secure web sockets</td>
<td>SSL certificate verification with Puppet CA</td>
<td>n/a</td>
</tr>
<tr>
<td>Service/Component</td>
<td>Communication Protocol</td>
<td>Authentication</td>
<td>Authorization</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>PCP broker</td>
<td>Secure web sockets</td>
<td>SSL certificate verification with Puppet CA</td>
<td>trapperkeeper-auth</td>
</tr>
<tr>
<td>File sync</td>
<td>HTTPS, SSL/TLS</td>
<td>SSL certificate verification with Puppet CA</td>
<td>trapperkeeper-auth</td>
</tr>
<tr>
<td>Code Manager</td>
<td>HTTPS; can fetch code remotely via HTTP, HTTPS, and SSH (via Git)</td>
<td>RBAC token authentication; for remote module sources, HTTP(S) Basic or SSH keys</td>
<td>RBAC user-based authorization; for remote module sources, HTTP(S) Basic or SSH keys</td>
</tr>
</tbody>
</table>

**Compatible ciphers**

PE is compatible with a variety of ciphers for different services.

**Note:** To use TLSv1.3, you must have both TLSv1.2 and TLSv1.3 enabled.

Most TLSv1.2 ciphers are available in either IANA or OpenSSL format, depending on the service its used for. The same TLSv1.3 ciphers can be used interchangeably for OpenSSL and IANA formats.

Use IANA format for these services on TLSv1.2:

- Puppet Server
- PuppetDB
- Console services
- Orchestrator

Use OpenSSL format for these services on TLSv1.2:

- Bolt Server
- ACE server
- PostgreSQL
- NGINX

**Note:** To use ECDSA ciphers, you must use your own CA certificates with ECC keys rather than Puppet Server generated certificates.

These are the default TLSv1.2 and TLSv1.3 ciphers accepted by PE for FIPS and non-FIPS installations. If you use a cipher that isn't listed, it is rejected when the service is establishing a connection.

<table>
<thead>
<tr>
<th>Ciphers in IANA format</th>
<th>Ciphers in OpenSSL format</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_AES_256_GCM_SHA384</td>
<td>TLS_AES_256_GCM_SHA384</td>
</tr>
<tr>
<td>TLS_AES_128_GCM_SHA256</td>
<td>TLS_AES_128_GCM_SHA256</td>
</tr>
<tr>
<td>TLS_CHACHA20_POLY1305_SHA256</td>
<td>TLS_CHACHA20_POLY1305_SHA256</td>
</tr>
</tbody>
</table>

**Note:** Bolt server, ACE server, and NGINX only

<table>
<thead>
<tr>
<th>Ciphers in OpenSSL format</th>
<th>Note: NGINX only</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDHE-ECDSA-CHACHA20-POLY1305_SHA256</td>
<td>Note: Bolt server, ACE server, and NGINX only</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
### Component versions in recent PE releases

These tables show which components are in Puppet Enterprise (PE) 2019.y and 2018.y. releases. Component version tables for prior releases are available in the Documentation for other PE versions on page 30. Component version tables for the current STS and LTS releases are available in the latest PE documentation.

#### Puppet Enterprise agent and server components

This table shows the components installed on all agent nodes.

**Note:** Hiera 5 is a backwards-compatible evolution of Hiera, which is built into Puppet 4.9.0 and higher. To provide some backwards-compatible features, it uses the classic Hiera 3.x.x codebase version listed in this table.

<table>
<thead>
<tr>
<th>PE Version</th>
<th>Puppet Agent</th>
<th>Puppet</th>
<th>Facter</th>
<th>Hiera</th>
<th>Resource API</th>
<th>MCollective Ruby</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.12 (overlap support)</td>
<td>6.28.0</td>
<td>6.28.0</td>
<td>3.14.24</td>
<td>3.9.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.11</td>
<td>6.27.1</td>
<td>6.27.0</td>
<td>3.14.23</td>
<td>3.9.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.10</td>
<td>6.26.0</td>
<td>6.26.0</td>
<td>3.14.22</td>
<td>3.8.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.9</td>
<td>6.25.1</td>
<td>6.25.1</td>
<td>3.14.21</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.8</td>
<td>6.24.0</td>
<td>6.24.0</td>
<td>3.14.19</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.7</td>
<td>6.23.0</td>
<td>6.23.0</td>
<td>3.14.18</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.6</td>
<td>6.22.1</td>
<td>6.22.1</td>
<td>3.14.17</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.5</td>
<td>6.21.1</td>
<td>6.21.1</td>
<td>3.14.16</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>2019.8.4</td>
<td>6.19.1</td>
<td>6.19.1</td>
<td>3.14.14</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>2019.8.3</td>
<td>6.19.1</td>
<td>6.19.1</td>
<td>3.14.14</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>2019.8.1</td>
<td>6.17.0</td>
<td>6.17.0</td>
<td>3.14.12</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>2019.8</td>
<td>6.16.0</td>
<td>6.16.0</td>
<td>3.14.11</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>2019.7</td>
<td>6.15.0</td>
<td>6.15.0</td>
<td>3.14.10</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
</tr>
<tr>
<td>PE Version</td>
<td>Puppet Agent</td>
<td>Puppet Server</td>
<td>Facter</td>
<td>PuppetDB</td>
<td>R10K</td>
<td>Razor Server</td>
<td>Bolt Services</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>2019.5</td>
<td>6.14.0</td>
<td>6.14.0</td>
<td>3.14.9</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.7</td>
</tr>
<tr>
<td>2019.4</td>
<td>6.14.0</td>
<td>6.14.0</td>
<td>3.14.9</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.7</td>
</tr>
<tr>
<td>2019.3</td>
<td>6.12.0</td>
<td>6.12.0</td>
<td>3.14.7</td>
<td>3.6.0</td>
<td>1.8.11</td>
<td>N/A</td>
<td>2.5.7</td>
</tr>
<tr>
<td>2018.1.18</td>
<td>5.5.22</td>
<td>5.5.22</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
</tr>
<tr>
<td>2018.1.17</td>
<td>5.5.22</td>
<td>5.5.22</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
</tr>
<tr>
<td>2018.1.16</td>
<td>5.5.21</td>
<td>5.5.21</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.5.20</td>
<td>5.5.20</td>
<td>3.11.13</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.5.19</td>
<td>5.5.19</td>
<td>3.11.12</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.5.18</td>
<td>5.5.18</td>
<td>3.11.11</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.5.17</td>
<td>5.5.17</td>
<td>3.11.10</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.5.16</td>
<td>5.5.16</td>
<td>3.11.9</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.5.14</td>
<td>5.5.14</td>
<td>3.11.8</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.5.10</td>
<td>5.5.10</td>
<td>3.11.7</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.5.8</td>
<td>5.5.8</td>
<td>3.11.6</td>
<td>3.4.5</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.4</td>
</tr>
<tr>
<td>2018.1.4</td>
<td>5.5.6</td>
<td>5.5.6</td>
<td>3.11.4</td>
<td>3.4.4</td>
<td>N/A</td>
<td>2.12.3</td>
<td>2.4.4</td>
</tr>
<tr>
<td>2018.1.3</td>
<td>5.5.4</td>
<td>5.5.3</td>
<td>3.11.3</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.2</td>
<td>2.4.4</td>
</tr>
<tr>
<td>2018.1.2</td>
<td>5.5.3</td>
<td>5.5.2</td>
<td>3.11.2</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.2</td>
<td>2.4.4</td>
</tr>
<tr>
<td>2018.1.0</td>
<td>5.5.1</td>
<td>5.5.1</td>
<td>3.11.1</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.1</td>
<td>2.4.4</td>
</tr>
</tbody>
</table>

This table shows components installed on server nodes.
Primary server and agent compatibility

Use this table to verify that you’re using a compatible version of the agent for your PE or Puppet server.

<table>
<thead>
<tr>
<th>PE Version</th>
<th>Puppet Server</th>
<th>PuppetDB/10k</th>
<th>Razor Server</th>
<th>Bolt Services</th>
<th>Agentless PostgreSQL Executor (ACE) Services</th>
<th>ActiveMQ</th>
<th>Nginx</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8</td>
<td>6.12.0</td>
<td>6.11.1</td>
<td>3.5.1</td>
<td>N/A</td>
<td>2.11.1</td>
<td>1.2.0</td>
<td>11.8</td>
</tr>
<tr>
<td>2019.7</td>
<td>6.11.1</td>
<td>6.10.1</td>
<td>3.5.0</td>
<td>1.9.7</td>
<td>2.6.0</td>
<td>1.1.1</td>
<td>11.7</td>
</tr>
<tr>
<td>2019.5</td>
<td>6.9.2</td>
<td>6.9.1</td>
<td>3.4.1</td>
<td>1.9.7</td>
<td>1.49.0</td>
<td>1.1.1</td>
<td>11.7</td>
</tr>
<tr>
<td>2019.4</td>
<td>6.9.1</td>
<td>6.9.1</td>
<td>3.4.1</td>
<td>1.9.7</td>
<td>1.49.0</td>
<td>1.1.1</td>
<td>11.7</td>
</tr>
<tr>
<td>2019.3</td>
<td>6.8.0</td>
<td>6.8.1</td>
<td>3.4.0</td>
<td>1.9.6</td>
<td>1.45.0</td>
<td>1.0.0</td>
<td>11.6</td>
</tr>
<tr>
<td>2018.1.18</td>
<td>5.3.16</td>
<td>5.2.19</td>
<td>2.6.9</td>
<td>1.9.9</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.20</td>
</tr>
<tr>
<td>2018.1.17</td>
<td>5.3.16</td>
<td>5.2.19</td>
<td>2.6.9</td>
<td>1.9.9</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.19</td>
</tr>
<tr>
<td>2018.1.16</td>
<td>5.3.14</td>
<td>5.2.18</td>
<td>2.6.8</td>
<td>1.9.9</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.18</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.3.13</td>
<td>5.2.15</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.17</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.3.12</td>
<td>5.2.13</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.17</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.3.11</td>
<td>5.2.12</td>
<td>2.6.7</td>
<td>1.9.6</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.16</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.3.10</td>
<td>5.2.11</td>
<td>2.6.7</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.15</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.3.9</td>
<td>5.2.9</td>
<td>2.6.6</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.13</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.3.8</td>
<td>5.2.8</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.12</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.3.7</td>
<td>5.2.7</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.3.6</td>
<td>5.2.6</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.4</td>
<td>5.3.5</td>
<td>5.2.4</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.3</td>
<td>5.3.4</td>
<td>5.2.4</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.2</td>
<td>5.3.3</td>
<td>5.2.2</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.0</td>
<td>5.3.2</td>
<td>5.2.2</td>
<td>2.6.2</td>
<td>1.8.1</td>
<td>N/A</td>
<td>N/A</td>
<td>9.6.8</td>
</tr>
</tbody>
</table>

Note: Puppet 5.x has reached end of life and is not actively developed or tested. We retain agent 5.x compatibility with later versions of the server only to enable upgrades.
Task compatibility

This table shows which version of the Puppet task specification is compatible with each version of PE.

<table>
<thead>
<tr>
<th>PE version</th>
<th>Puppet task specification (GitHub)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.0.1+</td>
<td>version 1, revision 3</td>
</tr>
<tr>
<td>2019.0.0+</td>
<td>version 1, revision 2</td>
</tr>
<tr>
<td>2017.3.0+</td>
<td>version 1, revision 1</td>
</tr>
</tbody>
</table>

FIPS 140-2 enabled PE

Puppet Enterprise (PE) is available in a FIPS (Federal Information Processing Standard) 140-2 enabled version. This version is compatible with select third party FIPS-compliant platforms.

To install FIPS-enabled PE, install the appropriate FIPS-enabled primary server or agent package on a supported platform with FIPS mode enabled. Primary and compiler nodes must be configured with sufficient available entropy for the installation process to succeed.

Changes in FIPS-enabled PE installations

In order to operate on FIPS-compliant platforms, PE includes the following changes:

- All components are built and packaged against system OpenSSL for the primary server, or against OpenSSL built in FIPS mode for agents.
- All use of MD5 hashes for security has been eliminated and replaced.
- Forge and module tooling use SHA-256 hashes to verify the identity of modules.
- Proper random number generation devices are used on all platforms.
- All Java and Clojure components use FIPS Bouncy Castle encryption providers on FIPS-compliant platforms.

Limitations and cautions for FIPS-enabled PE installations

Be aware of the following when installing FIPS-enabled PE.

- Migrating from non-FIPS versions of PE to FIPS-enabled PE requires reinstalling on a supported platform with FIPS mode enabled.
- Disaster recovery configurations are not supported for FIPS-enabled PE.
- FIPS-enabled PE installations don't support extensions or modules that use the standard Ruby OpenSSL library, such as hiera-eyaml. As a workaround, you can use a non-FIPS-enabled primary server with FIPS-enabled agents, which limits the issue to situations where only the primary uses the Ruby library. This limitation does not apply to versions 1.1.0 and later of the splunk_hec module, which supports FIPS-enabled servers. The FIPS Mode section of the module's Forge page explains the limitations of running this module in a FIPS environment.

Related information

Supported operating systems and devices on page 184
You can install PE and the agent on these supported platforms.

Cloud PE

Puppet Enterprise 2019.8 cloud images are available from several providers.

Cloud images contain a standard installation with PE services running on a primary server installed in your chosen cloud environment. You can use cloud images to manage deployments up to 2,500 nodes. Cloud deployments beyond 2,500 nodes aren't recommended because cloud PE doesn't support compilers.

After launching a cloud image, you can use PE just as you would any on-premises standard installation.
Use these tips and links as you work with your cloud deployment.

**Cloud providers**

PE cloud images are available from these providers:

- **AWS Marketplace**
- **Microsoft Azure Marketplace**
- **Oracle Cloud Marketplace**

**Licensing**

Cloud images follow a bring-your-own-license (BYOL) model, so you can use any existing PE license with your chosen cloud provider. With AWS, you may also purchase licenses directly from the AWS Marketplace using Enterprise Discount Program (EDP) credits. View [AWS pricing information](#) for details.

**System requirements**

Follow [hardware requirements for cloud deployments](#).

**Security groups**

Follow [firewall requirements for standard installations](#). See a sample security group policy in EC2-style JSON here.

**Identity and access management**

For AWS, follow Identity and Access Management (IAM) best practices to create the deployment user and role:

- [Security best practices in IAM](#)
- Grant least privilege
- [AWS account root user](#)

**Installing**

Follow instructions to [install PE in cloud environments](#).

**Connecting agents**

Cloud images contain agent packages for all supported platforms. You can [install agents](#) using any supported method.

To manage nodes outside of your cloud provider, or across cloud deployments, configure your primary server by running the `update_agent_repos.sh` script:

```
sudo /opt/puppetlabs/cloud/bin/update_agent_repos.sh public
```

**Configuring DNS**

In cloud deployments, PE uses the primary server's private hostname to generate certificates, and includes the public hostname and `puppet` as alternate DNS names.

Managing nodes by their private hostname retains consistency when nodes are resized or changed to a different image type.

**Tuning**

PE cloud images are tuned using default settings for standard installations. You can [tune your primary server](#) in cloud deployments as you would any on-premises installation.
Running commands on nodes

When running these commands on nodes in Azure environments, you must first switch to the superuser role by running `sudo su`.

- puppet agent -t
- puppet enterprise support
- puppet infrastructure
- puppet license
- puppet lookup
- puppet node
- puppet plugin
- puppetserver ca

Scaling

As your infrastructure grows, moving to a larger cloud instance or VM can improve system performance. Follow instructions from your cloud provider to scale your deployment.

- AWS – Resizing an EBS-backend instance

  **Note:** During EC2 resizing, your instance's public hostname and IP address might change. To access the PE console, connect to the new public hostname. Resizing the instance doesn't change the private hostname or IP address, and therefore no change is required for PE services or managed nodes.

- Azure – Add a data disk to a VM through the Portal

Upgrading

Follow instructions to upgrade a standard installation.

Troubleshooting

Find tips on troubleshooting cloud deployments.

Example EC2 security group policy

This JSON structure is an example EC2 security group policy that accommodates inbound network ports required by Puppet.

```json
{
  "IpPermissions": [
    {
      "PrefixListIds": [],
      "FromPort": 22,
      "IpRanges": ["CidrIp": "0.0.0.0/0"],
      "ToPort": 22,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 443,
      "IpRanges": ["CidrIp": "0.0.0.0/0"],
      "ToPort": 443,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 22,
      "IpRanges": ["CidrIp": "0.0.0.0/0"],
      "ToPort": 22,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    }
  ]
}
```
Getting support

You can get commercial support for versions of Puppet Enterprise (PE) in the leading-edge release stream (also known as STS), long-term support (LTS), and overlap support (extended support for prior LTS streams until EOL). You can also get support from our user community.

Puppet Enterprise support life cycle

Puppet Enterprise (PE) release streams are considered short-term support (STS), long-term support (LTS), overlap support (extended support until EOL), and end of life (EOL).

Note: STS is the leading-edge release stream, also called the Puppet Enterprise (PE) release track.

For full information about release types, support phases and dates for each release, release frequency, and upgrade recommendations, go to the Puppet Enterprise lifecycle policy page.

If the latest release with the most up-to-date features is right for you, download or try the latest PE release, or download an older supported release from the Previous Releases page. We recommend following our Installing on page 177 guide and understanding the System requirements on page 182 before downloading the installation package.

© 2022 Puppet, Inc., a Perforce company
Open source tools and libraries

PE uses open source tools and libraries. We use both externally maintained components, such as Ruby, PostgreSQL, and JVM, and projects we own and maintain, such as Facter, Puppet agent, Puppet Server, and PuppetDB.

Projects we own and maintain are “upstream” of our commercial releases. Our open source projects move faster and have shorter support life cycles than PE. We might discontinue updates to our open source platform components before their commercial EOL dates. We vet upstream security and feature releases and update supported versions according to customer demand and our Security policy.

Support portal

We provide responsive, dependable, quality support to resolve any issues regarding the installation, operation, and use of Puppet Enterprise (PE).

PE has two commercial support plans: Standard and Premium. Both allow you to report your support issues to our confidential customer support portal. When you purchase PE, you receive an account and login details for the portal, which includes access to our knowledge base.

Note: The term standard installation refers to a PE installation with up to 4,000 nodes. The Standard Support Plan is not limited to this installation type. In the support context, Standard refers to the support level, not the PE installation size.

Puppet metrics collector

The Puppet metrics collector can help troubleshoot performance issues with Puppet Enterprise (PE) components.

Puppet Professional Services and Support use and recommend the tool to help optimize PE installations.

The Puppet metrics collector is packaged in a module that is installed with PE. By default, the module is disabled. You can enable and disable metrics collection by setting Boolean values for these parameters:

- puppet_enterprise::enable_metrics_collection (for Puppet services metrics)
- puppet_enterprise::enable_system_metrics_collection (for system metrics)

For information and instructions, refer to Puppet Enterprise metrics on page 414.

Important: If you have a version of the puppetlabs-puppet_metrics_collector module, from the Forge or other sources, specified in the code, you must remove this version before upgrading to allow the version bundled with PE to be asserted.

Related information

Methods for configuring PE on page 286
After you’ve installed Puppet Enterprise (PE), optimize it for your environment by configuring and tuning settings as needed. For example, you might want to add your certificate to the allowlist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

Puppet Enterprise metrics on page 414
You can use Puppet Enterprise (PE) metrics for your own performance tuning or provide them to Support for troubleshooting.

PE support script

When seeking support, you might be asked to run an information-gathering support script. This script collects a large amount of system information and Puppet Enterprise (PE) diagnostics, compresses the data, and prints the location of the zipped tarball when it finishes running.

The pe_support_script module, bundled with the installer, provides the script.
Running the support script

Run the support script on the command line of your primary server or any agent node running Red Hat Enterprise Linux, Ubuntu, or SUSE Linux Enterprise Server operating systems with the command: `/opt/puppetlabs/bin/puppet enterprise support`

PE version 2019.8.9 includes version 3 of the support script. Version 3 has more options that can be used to modify the support script behavior. As such, some options in version 3 are not available in combination with the `--v1` option. This is because the `--v1` option activates the legacy (version 1) support script. Options not compatible with version 1 are designated in the table below as *Not compatible with the `--v1` parameter*.

These options can modify the support script output:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--verbose</code></td>
<td>Logs verbosely.</td>
</tr>
<tr>
<td><code>--debug</code></td>
<td>Logs debug information.</td>
</tr>
<tr>
<td><code>--classifier</code></td>
<td>Collects classification data.</td>
</tr>
<tr>
<td><code>--dir &lt;DIRECTORY&gt;</code></td>
<td>Specifies where to save the support script's resulting tarball.</td>
</tr>
<tr>
<td><code>--ticket &lt;NUMBER&gt;</code></td>
<td>Specifies a support ticket number for record-keeping purposes.</td>
</tr>
<tr>
<td><code>--encrypt</code></td>
<td>Encrypts the support script's resulting tarball with GnuPG encryption.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>You must have GPG or GPG2 available in your PATH in order to encrypt the tarball.</td>
</tr>
<tr>
<td><code>--log_age</code></td>
<td>Specifies how many days' worth of logs the support script collects. Valid values are positive integers or <code>all</code> to collect all logs, up to 1 GB per log. Default is 7 (seven days).</td>
</tr>
<tr>
<td><code>--v1</code></td>
<td>Activate version 1 of the support script. This option is not compatible with options designated as <em>Not compatible with the <code>--v1</code> parameter</em>.</td>
</tr>
<tr>
<td><code>--v3</code></td>
<td>Activate version 3 of the support script. This is a default but can be overridden with <code>--v1</code>.</td>
</tr>
<tr>
<td><code>--list</code></td>
<td>List diagnostics that may be enabled or disabled. Diagnostic labeled &quot;opt-in&quot; must be explicitly enabled. All others are enabled by default. <em>Not compatible with the <code>--v1</code> parameter</em>.</td>
</tr>
<tr>
<td><code>--enable &lt;LIST&gt;</code></td>
<td>A comma-separated list of diagnostic names to enable. Use the <code>--list</code> option to print available names. The <code>--enable</code> option must be used to activate diagnostics marked as &quot;opt-in.&quot; <em>Not compatible with the <code>--v1</code> parameter</em>.</td>
</tr>
<tr>
<td><code>--disable &lt;LIST&gt;</code></td>
<td>A comma-separated list of diagnostic names to disable. Use the <code>--list</code> option to print available names. <em>Not compatible with the <code>--v1</code> parameter</em>.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--only &lt;LIST&gt;</code></td>
<td>A comma-separated list of diagnostic names to enable. All other diagnostics are disabled. Use the <code>--list</code> option to print available names. <em>Not compatible with the <code>--v1</code> parameter.</em></td>
</tr>
<tr>
<td><code>--upload</code></td>
<td>Upload the output tarball to Puppet Support via SFTP. Requires the <code>--ticket &lt;NUMBER&gt;</code> option to be used. <em>Not compatible with the <code>--v1</code> parameter.</em></td>
</tr>
<tr>
<td><code>--upload_disable_host_key_check</code></td>
<td>Disable SFTP host key checking. Go to Use SFTP to upload files to Puppet Support for a list of current host key values. <em>Not compatible with the <code>--v1</code> parameter.</em></td>
</tr>
<tr>
<td><code>--upload_user &lt;USER&gt;</code></td>
<td>Specify a SFTP user to use when uploading. If not specified, a shared write-only account is used. <em>Not compatible with the <code>--v1</code> parameter.</em></td>
</tr>
<tr>
<td><code>--upload_key &lt;FILE&gt;</code></td>
<td>Specify a SFTP key to use with <code>--upload_user</code>. <em>Not compatible with the <code>--v1</code> parameter.</em></td>
</tr>
</tbody>
</table>

These code examples show how to use options when running the support script:

```bash
# Collect diagnostics for just Puppet agent and Puppet Server
/opt/puppetlabs/bin/puppet enterprise support --only puppet-agent,puppetserver

# Enable collection of PE classification
/opt/puppetlabs/bin/puppet enterprise support --enable pe.console.classifier-groups

# Disable collection of system logs, upload result to Puppet Support via SFTP
/opt/puppetlabs/bin/puppet enterprise support --disable system.logs --upload --ticket 12345
```

Descriptions of diagnostics you can select with the `--enable`, `--disable`, and `--only` flags are in the next section.

**Information collected by the support script**

This information is collected by the support script.

**base-status**

The base-status check collects basic diagnostics about the PE installation. This check is enabled and is not affected by the `--disable` or `--only` flags.

Specifically, the base-status check collects the support script version, the Puppet ticket number (if supplied), and the time the script ran.

**system**

The checks in the system scope gather diagnostics, logs, and configuration related to the operating system.

The system.config check collects:

- A copy of `/etc/hosts`
- A copy of `/etc/nsswitch.conf`
- A copy of `/etc/resolv.conf`
- Configuration for the APT, YUM, and dnf package managers
- The operating system version
• The umask in effect
• The status of SELinux
• A list of configured network interfaces
• A list of configured firewall rules
• A list of loaded firewall kernel modules

The `system.logs` check collects copies of the system log (syslog) and kernel log (dmesg).

The `system.status` check collects:
• Values of variables set in the environment
• A list of running processes
• A list of enabled services
• A list of systemd timers
• The uptime of the system
• A list of established network connections
• NTP status
• The IP address and hostname of the node running the script, according to DNS
• Disk usage
• RAM usage

**puppet-agent**

The checks in the puppet-agent scope gather diagnostics, logs, and configuration related to the Puppet agent services.

The `puppet.config` check collects:
• Facter configuration files from `/etc/puppetlabs/facter/facter.conf`
• Puppet configuration files from `/etc/puppetlabs/puppet/device.conf`, `/etc/puppetlabs/puppet/hiera.yaml`, and `/etc/puppetlabs/puppet/puppet.conf`
• PXP agent configuration files from `/etc/puppetlabs/pxp-agent/modules/` and `/etc/puppetlabs/pxp-agent/pxp-agent.conf`

The `puppet-agent.logs` check collects:
• Puppet log files from `/var/log/puppetlabs/puppet`
• JournalD logs for the puppet service and pxp-agent service
• PXP agent log files from `/var/log/puppetlabs/puppet`

The `puppet-agent.status` check collects:
• `facter -p` output and debug-level messages
• A list of Ruby gems installed for use by Puppet
• Ping output for the Puppet Server the agent is configured to use
• A copy of the graphs/ directory and the classes.txt and last_run_summary.yaml files from the Puppet statedir
• A listing of metadata (name, size, etc.) of files present in the `/etc/puppetlabs`, `/var/log/puppetlabs`, and `/opt/puppetlabs` directories.
• A listing of Puppet and PE packages installed on the system along with verification output for each

**puppetserver**

The checks in the puppetserver scope gather diagnostics, logs, and configuration related to the Puppet Server service.

The `puppetserver.config` check collects these Puppet Server configuration files:
• `/etc/puppetlabs/code/hiera.yaml`
• `/etc/puppetlabs/puppet/auth.conf`
• `/etc/puppetlabs/puppet/autosign.conf`

© 2022 Puppet, Inc., a Perforce company
• /etc/puppetlabs/puppet/classifier.yaml
• /etc/puppetlabs/puppet/fileserver.conf
• /etc/puppetlabs/puppet/hiera.yaml
• /etc/puppetlabs/puppet/puppet.conf
• /etc/puppetlabs/puppet/puppetdb.conf
• /etc/puppetlabs/puppet/routes.yaml
• /etc/puppetlabs/puppetserver/bootstrap.cfg
• /etc/puppetlabs/puppetserver/code-manager-request-logging.xml
• /etc/puppetlabs/puppetserver/conf.d/
• /etc/puppetlabs/puppetserver/logback.xml
• /etc/puppetlabs/puppetserver/request-logging.xml
• /etc/puppetlabs/puppetserver/r10k/r10k.yaml
• /opt/puppetlabs/server/data/code-manager/r10k.yaml

The puppetserver.logs check collects:
• Puppet Server log files from /var/log/puppetlabs/puppetserver/
• JournalD logs for the pe-puppetserver service
• r10k log files from /var/log/puppetlabs/r10k/

The puppetserver.metrics check collects data stored in /opt/puppetlabs/puppet-metrics-collector/puppetserver

The puppetserver.status check collects:
• A list of certificates issued by the Puppet CA
• A list of Ruby gems installed for use by Puppet Server
• Output from the status/v1/services API
• Output from the puppet/v3/environment_modules API
• Output from the analytics/v1/collections/snapshots API
• Output from the puppet/v3/environments API
• environment.conf and hiera.yaml files from each Puppet code environment
• The disk space used by Code Manager cache, storage, client, and staging directories
• The disk space used by the server's File Bucket
• The output of r10k deploy display

puppetdb

The checks in the puppetdb scope gather diagnostics, logs, and configuration related to the PuppetDB service.

The puppetdb.config check collects these configuration files:
• /etc/puppetlabs/puppetdb/bootstrap.cfg
• /etc/puppetlabs/puppetdb/certificate-whitelist
• /etc/puppetlabs/puppetdb/conf.d/
• /etc/puppetlabs/puppetdb/logback.xml
• /etc/puppetlabs/puppetdb/request-logging.xml

The puppetdb.logs check collects PuppetDB log files (/var/log/puppetlabs/puppetdb) and JournalD logs for the pe-puppetdb service.

The puppetdb.metrics check collects data stored in /opt/puppetlabs/puppet-metrics-collector/puppetdb.

The puppetdb.status check collects:
• Output from the status/v1/services API
• Output from the pdb/admin/v1/summary-stats API
• A list of active certnames from the PQL query nodes[certname] {deactivated is null and expired is null}

pe
The checks in the pe scope gather diagnostics, logs, and configuration related to Puppet Enterprise services.
The pe.config check collects:
• Installer configuration files:
  • /etc/puppetlabs/enterprise/conf.d/
  • /etc/puppetlabs/enterprise/hiera.yaml
  • /etc/puppetlabs/installer/answers.install
• PE client tools configuration files:
  • /etc/puppetlabs/client-tools/orchestrator.conf
  • /etc/puppetlabs/client-tools/puppet-access.conf
  • /etc/puppetlabs/client-tools/puppet-code.conf
  • /etc/puppetlabs/client-tools/puppetdb.conf
  • /etc/puppetlabs/client-tools/services.conf
The pe.logs check collects:
• PE installer log files from /var/log/puppetlabs/installer/
• PE backup and restore log files from /var/log/puppetlabs/pe-backup-tools/ and /var/log/
puppetlabs/puppet_infra_recover_config_cron.log
The pe.status check collects output from puppet infra status, current tuning settings from puppet infra tune, and recommended tuning settings from puppet infra tune.
The pe.file-sync check is disabled by default. When activated by the --enable option, this check collects:
• Puppet manifests and other content from /etc/puppetlabs/code-staging/
• Puppet manifests and other content stored in Git repos under /opt/puppetlabs/server/data/
puppetserver/filesync

pe.console
The checks in the pe.console scope gather diagnostics, logs, and configuration related to the Puppet Enterprise console service.
The pe.console.config check collects these configuration files:
• /etc/puppetlabs/console-services/bootstrap.cfg
• /etc/puppetlabs/console-services/conf.d/
• /etc/puppetlabs/console-services/logback.xml
• /etc/puppetlabs/console-services/rbac-certificate-whitelist
• /etc/puppetlabs/console-services/request-logging.xml
• /etc/puppetlabs/nginx/conf.d/
• /etc/puppetlabs/nginx/nginx.conf
The pe.console.logs check collects:
• Console log files from /var/log/puppetlabs/console-services/ and /var/log/
puppetlabs/nginx/
• JournalD logs for the pe-puppetdb and pe-nginx services
The pe.console.status check collects:
• Output from the /status/v1/services API
• Directory service connection configuration, with passwords removed
The `pe.console.classifier-groups` check is disabled by default. When activated by the `--enable` option, this check collects all classification data provided by the `/v1/groups` API endpoint.

**pe.orchestration**

The checks in the `pe.orchestration` scope gather diagnostics, logs, and configuration related to the Puppet Enterprise orchestration services.

The `pe.orchestration.config` check collects:

- ACE server configuration files from `/etc/puppetlabs/puppet/ace-server/conf.d/`
- Bolt server configuration files from `/etc/puppetlabs/puppet/bolt-server/conf.d/`
- Orchestration service configuration files:
  - `/etc/puppetlabs/puppet/orchestration-services/bootstrap.cfg`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/analytics.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/auth.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/global.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/inventory.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/metrics.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/orchestrator.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/pcp-broker.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/web-routes.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/conf.d/webserver.conf`
  - `/etc/puppetlabs/puppet/orchestration-services/logback.xml`
  - `/etc/puppetlabs/puppet/orchestration-services/request-logging.xml`

The `pe.orchestration.logs` check collects:

- ACE server log files from `/var/log/puppetlabs/ace-server/`
- Bolt server log files from `/var/log/puppetlabs/bolt-server/`
- Orchestrator log files from `/var/log/puppetlabs/orchestration-services/`
- JournalD logs for the `pe-ace-server` service, `pe-bolt-server` service, and `pe-orchestration-services` service

The `pe.orchestration.metrics` check collects data stored in `/opt/puppetlabs/puppet-metrics-collector/orchestrator/`.

The `pe.orchestration.status` check collects output from the `/status/v1/services` API.

**pe.postgres**

The checks in the `pe.postgres` scope gather diagnostics, logs, and configuration related to the Puppet Enterprise PostgreSQL database.

The `pe.postgres.config` check collects these configuration files:

- `/opt/puppetlabs/server/data/postgresql/*/data/postgresql.conf`
- `/opt/puppetlabs/server/data/postgresql/*/data/postmaster.opts`
- `/opt/puppetlabs/server/data/postgresql/*/data/pg_ident.conf`
- `/opt/puppetlabs/server/data/postgresql/*/data/pg_hba.conf`

The `pe.postgres.logs` check collects JournalD logs for the `pe-postgresql` service and these PostgreSQL log files:

- `/var/log/puppetlabs/postgresql/*/`
- `/opt/puppetlabs/server/data/postgresql/pg_upgrade_internal.log`
- `/opt/puppetlabs/server/data/postgresql/pg_upgrade_server.log`
- `/opt/puppetlabs/server/data/postgresql/pg_upgrade_utility.log`

The `pe.postgres.status` check collects:
• A list of setting values that the database is using while running
• A list of currently established database connections and the queries being executed
• A distribution of Puppet run start times for thundering herd detection
• The status of any configured replication slots
• The status of any active replication connections
• The size of database directories on disk
• The size of databases as reported by the database service
• The size of tables and indices within databases

Community support
As a Puppet Enterprise (PE) customer, you are welcome to participate in our large and helpful open source community as well as report issues against the open source project.

• Join the Puppet Enterprise Users group. Your request to join is sent to Puppet by Perforce for authorization, and you receive an email when you’ve been added to the user group.
  • Click “Sign in and apply for membership.”
  • Click “Enter your email address to access the document.”
  • Enter your email address.
• Join the open source Puppet Users group.
• Join the Puppet Developers group.
• Report issues with the open source Puppet project.

Using the PE docs
Review these tips to get the most out of the PE docs.

Using example commands
These guidelines can help you understand and customize example commands included in the Puppet Enterprise (PE) docs.

puppet commands generate cURL arguments
Some examples in the PE docs use puppet commands to populate some cURL arguments and take the guesswork out of providing those values. For example:

```
url="http://$(puppet config print server):4433"
curl "$url"
```

puppet commands can return different values depending on various conditions. To use cURL examples successfully, run the entire example (including setting the environment variables and the curl command) as root, Administrator, or with equivalent elevated privileges.

To run commands on a machine without elevated privileges, replace the inline puppet commands with hard-coded values. If you’re unsure about the correct values, run the puppet commands to get reasonable default values.

Authentication tokens in cURL commands
If a curl command requires authentication, the example might contain this line:

```
auth_header="X-Authentication: $(puppet-access show)"
```
If you have an actual authentication token available, you can use that in the command instead, such as:

```plaintext
auth_header="X-Authentication: <TOKEN>"
```

**Modifications for Windows**

While the commands in the PE docs are primarily *nix-based, Windows-specific commands are provided in topics focusing exclusively on Windows systems.

There are various options for running `curl` commands directly in Windows, such as:

- Installing the curl executable for Windows.
- Using built-in curl functionality included with Git for Windows.
- Using the GNU Bash shell.

If you're using PowerShell, you can use these equivalent commands to modify *nix `curl` commands for use in Windows:

<table>
<thead>
<tr>
<th>Native curl</th>
<th>PowerShell equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>curl</code></td>
<td><code>Invoke-WebRequest</code></td>
</tr>
<tr>
<td><code>-k</code> or <code>--insecure</code></td>
<td><code>[System.Net.ServicePointManager]::ServerCertificateValidationCallback = $true</code></td>
</tr>
<tr>
<td><code>-H</code></td>
<td><code>-Headers</code></td>
</tr>
<tr>
<td><code>-X</code></td>
<td><code>-Method</code></td>
</tr>
<tr>
<td><code>-d</code></td>
<td><code>-Body</code></td>
</tr>
<tr>
<td>\</td>
<td><code>\</code></td>
</tr>
</tbody>
</table>

You can learn more about `Invoke-WebRequest` and the arguments it accepts in the Microsoft [PowerShell documentation](https://learn.microsoft.com/en-us/powershell/module/system.net.webrequest).

**Commands with elevated privileges**

Some commands in PE require elevated privileges. Depending on the operating system, you can use either `sudo`, `runas`, or a root or admin user.

Elevated privileges allow you to access and do more than you might be able to with your personal account privileges. There are three primary methods for using elevated privileges:

**root (or administrator)**

In *nix systems, the root user has virtually unlimited access to read, write, or change files and system configurations; install, uninstall, and upgrade software; or perform any operation as any user. The equivalent in Windows is the administrator.

**sudo**

The `sudo` command, which means *super user do*, allows a user to execute a command from a personal user account with temporarily elevated privileges. With `sudo`, you can do most of the things the root user can do without actually logging in as the root user.

**Run as administrator or runas**

Using the `runas` command or running a program as an administrator (for example, by right-clicking the program and selecting **Run as administrator**) is the Windows equivalent of `sudo` – It allows you to temporarily perform administrator functions without actually logging in as the administrator.

You can use `sudo` to run almost all commands in Puppet with the exception of `puppet infrastructure` commands, which require you to be logged in as the root user (or administrator). You can run `puppet infrastructure help <ACTION>` to get information about `puppet infrastructure` commands.
**Restriction:** You must log in as the root user (or administrator) to run puppet infrastructure commands.

In Windows systems, use `runas` or open the command prompt as an administrator (recommended for PowerShell commands) instead of using `sudo`.

**Documentation for other PE versions**

Documentation for each PE version is initially published on our documentation website (where you are now). We actively maintain documentation for our leading-edge PE release stream (also known as STS), the current LTS stream, and, when applicable, the ongoing support stream (which is the previous LTS until it reaches EOL).

Documentation for end-of-life (EOL) and superseded major versions (formatted as `<YEAR>.y`, such as 2023.0, 2023.1, and so on) may continue to be available on our documentation website while no longer being updated, and, eventually, moved to our PE docs archive on GitHub.

For LTS releases, we do not separately publish documentation for each incremental version (formatted as `<YEAR>.y.z`, such as 2021.7.0, 2021.7.1, and so on). To find PDFs of prior LTS incremental versions, go to our PE docs archive on GitHub.

When we start a new LTS stream, we continue to host (but do not update) the prior major versions for that stream for some time. For example, if the LTS is 2021.7.z, then we retain 2021.0 through 2021.6 for a limited amount of time. For the prior LTS, we continue to host the latest increment of that stream during the overlap support period and up to one year after.

To find documentation for any version earlier than the current LTS stream’s earliest major version (such as 2021.0) or, when applicable, the most recent overlap support incremental version, go to our PE docs archive on GitHub.

Archived documentation is commonly retained as PDF. You may find some older versions retained in markdown format.

This table describes where you can find documentation for various PE versions and release streams:

<table>
<thead>
<tr>
<th>PE Version</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.z (overlap support)</td>
<td>You're currently viewing the documentation for the most-recent incremental release of our overlap support stream. For earlier incremental releases, go to the PE docs archive on GitHub.</td>
</tr>
<tr>
<td>Earlier versions</td>
<td>PE docs archive on GitHub</td>
</tr>
</tbody>
</table>

**Puppet platform documentation for PE**

Puppet Enterprise (PE) is built on the Puppet platform which has several components: Puppet, Puppet Server, Facter, Hiera, and PuppetDB. This page describes each of these platform components, and links to the component docs.

**Puppet**

- Puppet docs

Puppet is the core of our configuration management platform. It consists of a programming language for describing desired system states, an agent that can enforce desired states, and several other tools and services.

Right now, you’re reading the PE manual; the Puppet reference manual is a separate section of our docs site. After you’ve followed a link there, you can use the navigation sidebar to browse other sections of the manual.

© 2022 Puppet, Inc., a Perforce company
Note: The Puppet manual has information about installing the open source release of Puppet. As a PE user, ignore those pages.

The following pages are good starting points for getting familiar with Puppet:

**Language**
- An outline of how the Puppet language works.
- Resources, variables, conditional statements, and relationships and ordering are the fundamental pieces of the Puppet language.
- Classes and defined types are how you organize Puppet code into useful chunks. Classes are the main unit of Puppet code you’ll be interacting with on a daily basis. You can assign classes to nodes in the PE console.
- Facts and built-in variables explains the special variables you can use in your Puppet manifests.

**Modules**
- Most Puppet code goes in modules. We explain how modules work here.
- There are also guides to installing modules and publishing modules on the Forge.
- Use the code management features included in PE to control your modules instead of installing by hand. See Managing and deploying Puppet code (in the PE manual) for more details.

**Services and commands**
- An overview of Puppet’s architecture.
- A list of the main services and commands you’ll interact with.
- Notes on running Puppet’s commands on Windows.

**Built-in resource types and functions**
- The resource type reference has info about all of the built-in Puppet resource types.
- The function reference does the same for the built-in functions.

**Important directories and files**
- Most of your Puppet content goes in environments. Find out more about environments here.
- The codedir contains code and data and the confdir contains config files. The modulepath and the main manifest both depend on the current environment.

**Configuration**
- The main config file for Puppet is /etc/puppetlabs/puppet/puppet.conf. Learn more about Puppet’s settings, and about puppet.conf itself.
- There are also a bunch of other config files used for special purposes. Go to the page about puppet.conf and check the navigation sidebar for a full list.

**Puppet Server**
- Puppet Server docs

Puppet Server is the JVM application that provides the core Puppet HTTPS services. Whenever Puppet agent checks in to request a configuration catalog for a node, it contacts Puppet Server.

For the most part, PE users don’t need to directly manage Puppet Server, and the Puppet reference manual (above) has all the important info about how Puppet Server evaluates the Puppet language and loads environments and modules. However, some users might need to access the environment cache and JRuby pool administrative APIs, and there’s lots of interesting background information in the rest of the Puppet Server docs.

Note: The Puppet Server manual has information about installing the open source release of Puppet Server. As a PE user, ignore those pages. Additionally, the Puppet Server config files in PE are managed with a built-in Puppet module; to change most settings, set the appropriate class parameters in the console.
Facter

- **Facter docs**

Facter is a system profiling tool. Puppet agent uses it to send important system info to Puppet Server, which can access that info when compiling that node’s catalog.

- For a list of variables you can use in your code, check out the [core facts reference](https://docs.puppet.com/facter/).
- You can also write your own custom facts. See the [custom fact overview](https://docs.puppet.com/facter/) and the [custom fact walkthrough](https://docs.puppet.com/facter/).

Hiera

- **Hiera docs**

Hiera is a hierarchical data lookup tool. You can use it to configure your Puppet classes.

Start with the [overview](https://docs.puppet.com/hiera/) and use the navigation sidebar to get around.

**Note:** Hiera 5 is a backwards-compatible evolution of Hiera, which is built into Puppet. To provide some backwards-compatible features, it uses the classic Hiera 3 codebase. This means “Hiera” is still version 3.x, even though this Puppet Enterprise version uses Hiera 5.

PuppetDB

- **PuppetDB docs**

PuppetDB collects the data Puppet generates, and offers a powerful query API for analyzing that data. It’s the foundation of the PE console, and you can also use the API to build your own applications.

If you’re interacting with PuppetDB directly, you’ll mostly be using the query API.

- The [query tutorial page](https://docs.puppet.com/puppetdb/) walks you through the process of building and executing a query.
- The [query structure page](https://docs.puppet.com/puppetdb/) explains the fundamentals of using the query API.
- The [API curl tips page](https://docs.puppet.com/puppetdb/api/) has useful information about testing the API from the command line.
- You can use the navigation sidebar to browse the rest of the query API docs.

**Note:** The PuppetDB manual has information about installing the open source release of PuppetDB. As a PE user, ignore those pages.

**Related information**

Managing and deploying Puppet code on page 763

Puppet Enterprise (PE) includes built-in tools for managing and deploying your Puppet code. Code Manager and r10k are code management tools that automatically install modules, create and maintain environments, and deploy new code to your primary server and compilers, all based on version control of your Puppet code and data.

**API index**

APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Puppet Enterprise APIs**

For information on port requirements, see [System configuration](https://docs.puppet.com/puppetdb/) on page 190.
<table>
<thead>
<tr>
<th>API</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node inventory API on page 581</td>
<td>• Making HTTP(S) requests to the Puppet inventory service API.</td>
</tr>
<tr>
<td></td>
<td>• Creating and deleting connection entries in the inventory service database.</td>
</tr>
<tr>
<td></td>
<td>• Listing the connections entries in the inventory database.</td>
</tr>
<tr>
<td>RBAC service API v1</td>
<td>• Managing access to Puppet Enterprise.</td>
</tr>
<tr>
<td></td>
<td>• Connecting to external directories.</td>
</tr>
<tr>
<td></td>
<td>• Generating authentication tokens.</td>
</tr>
<tr>
<td></td>
<td>• Managing users, user roles, user groups, and user permissions.</td>
</tr>
<tr>
<td>RBAC service API v2</td>
<td>• Revoking authentication tokens.</td>
</tr>
<tr>
<td>Node classifier service API</td>
<td>• Querying the groups that a node matches.</td>
</tr>
<tr>
<td></td>
<td>• Querying the classes, parameters, and variables that have been assigned to a node or group.</td>
</tr>
<tr>
<td></td>
<td>• Querying the environment that a node is in.</td>
</tr>
<tr>
<td>Orchestrator API</td>
<td>• Gathering details about the orchestrator jobs you run.</td>
</tr>
<tr>
<td></td>
<td>• Inspecting applications and applications instances in your Puppet environments.</td>
</tr>
<tr>
<td>Code Manager API</td>
<td>• Creating a webhook to trigger Code Manager.</td>
</tr>
<tr>
<td></td>
<td>• Queueing Puppet code deployments.</td>
</tr>
<tr>
<td></td>
<td>• Checking Code Manager and file sync status.</td>
</tr>
<tr>
<td>Status API</td>
<td>• Checking the health status of PE services.</td>
</tr>
<tr>
<td>Activity service API</td>
<td>• Querying PE service and user events logged by the activity service.</td>
</tr>
<tr>
<td>Value API</td>
<td>• Generating reports about time and money freed by PE automation.</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
## Open source Puppet Server, Puppet, PuppetDB, and Forge APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Useful for</th>
</tr>
</thead>
</table>
| **Puppet Server administrative API endpoints** | • Deleting environment caches created by a primary server.  
• Deleting the Puppet Server pool of JRuby instances. |
| • **environment-cache** |  |
| • **jruby-pool** |  |
| **Server-specific Puppet API** | • Getting the classes and parameter information that is associated with an environment, with cache support.  
• Getting information about what modules are installed in an environment.  
• Getting the contents of a specific version of a file in a specific environment. |
| • **Environment classes** |  |
| • **Environment modules** |  |
| • **Static file content** |  |
| **Puppet Server status API** | • Checking the state, memory usage, and uptime of the services running on Puppet Server. |
| **Puppet Server metrics API** | • Querying Puppet Server performance and usage metrics. |
| • **v1 metrics** (deprecated) |  |
| • **v2 metrics** (Jolokia) |  |
| **Puppet HTTP API** | • Retrieving a catalog for a node.  
• Accessing environment information. |
| **Certificate Authority (CA) API** | • Used internally by Puppet to manage agent certificates. |
| **PuppetDB APIs** | • Querying the data that PuppetDB collects from Puppet.  
• Importing and exporting PuppetDB archives.  
• Changing the PuppetDB model of a population.  
• Querying information about the PuppetDB server.  
• Querying PuppetDB metrics. |
| **Forge API** | • Finding information about modules and users on the Forge.  
• Writing scripts and tools that interact with the Forge website. |
Release notes

These release notes contain important information about Puppet Enterprise® 2019.8.

This release incorporates new features, enhancements, and resolved issues from all previous major releases. If you're upgrading from an earlier version of PE, check the release notes for any interim versions for details about additional improvements in this release over your current release.

**Note:** This version of documentation represents the latest update in this release stream. There might be differences in features or functionality from previous releases in this stream.


- **PE release notes** on page 35
  These are the new features, enhancements, resolved issues, and deprecations in this version of PE.
- **PE known issues** on page 63
  These are the known issues in PE 2019.8.
- **What's new since PE 2018.1** on page 67
  These are the major new features, enhancements, deprecations, and removals since the previous LTS release (2018.y). Most bug fixes were ported back to PE 2018.1 if they affected that version, so there aren't any notable resolved issues between LTS versions.
- **Platform release notes** on page 73
  PE uses certain components of open source Puppet. Applicable platform release notes are collected here for convenience.

PE release notes

These are the new features, enhancements, resolved issues, and deprecations in this version of PE.


**PE 2019.8.12**

Released August 2022

**Important:** With this release, PE 2021.7 is our new PE LTS, and PE 2019.8.z is in overlap support until EOL in February 2023. For important information about upgrading from 2019.y to the new LTS, go to [Upgrading Puppet Enterprise in the 2021.7 documentation](https://puppet.com/docs/upgrading-puppet-enterprise-in-the-2021.7-documentation) and learn What’s new since 2019.8.

**Enhancements**

**Orchestrator API endpoints return "total": 0 if there are no jobs**

Orchestrator API v1 endpoints on page 695 that return pagination containing the total number of jobs (such as [GET /jobs on page 723], [GET /scheduled_jobs on page 735], and [GET /plan_jobs on page 743]) now return "total": 0, instead of "total": null, when there are no jobs.

**Addressed CVEs**

We updated the PostgreSQL driver in some PE component to address CVE-2022-31197. The application was not vulnerable to exploit prior to this update.

We also made changes to address CVE-2022-1292 and CVE-2022-2068.
Platform support

Ubuntu 16.04 is no longer a supported agent platform.

This version adds support for these platforms:

Agent
macOS 12 M1
Ubuntu (General Availability kernels) 22.04 x86_64
Microsoft Windows 11 x64

Client tools
Ubuntu (General Availability kernels) 22.04 x86_64
macOS 12 M1, M2

Patch Management
Ubuntu (General Availability kernels) 22.04 x86_64
Microsoft Windows 11 x64

Deprecations and removals

Ubuntu 16.04 is no longer a supported agent platform.

Resolved issues

full-deploy didn't override --incremental

Code Manager's full-deploy option, used for Configuring module deployment scope on page 785, now correctly overrides the default --incremental deploy behavior.

Code Manager couldn't fetch code on FIPS platforms

On FIPS platforms running PE versions 2019.8.10 or 2019.8.11, Code Manager and r10k couldn't fetch code from your code repo due to libssh attempting to use algorithms that are not allowed on FIPS. In PE 2019.8.12, the disallowed algorithms are disabled in libssh, allowing Code Manager and r10k to successfully fetch code.

Orchestrator ignored _noop when passed to run_task() through a plan

When a plan passed the _noop flag to the run_task() function, the PE Orchestrator now correctly acknowledges the _noop flag.

Orchestrator doesn't restart unexpectedly during the convert_legacy_compiler plan

Previously, when running the enterprise_tasks::convert_legacy_compiler plan, the hosts in the pcp-brokers array could change order. This caused the pe-orchestration-services service to restart (as a result of detecting a presumed configuration change) and, ultimately, caused the plan to fail.

Orchestrator couldn't run tasks within modules named tasks or scripts

You can now successfully run tasks that are within modules named tasks or scripts.

Incorrect run-time for splayed agent runs

In previous PE versions, when agent runs were splayed, the run-time reported in the PE console was incorrect.

Sensitive parameters sometimes exposed in cleartext in job results

Sensitive plan parameters from Bolt plans that execute actions over PCP transport are no longer stored in the orchestrator database and, therefore, are properly masked in the job results.
New features

Lockless code deploys are no longer experimental
Since debuting as an experimental feature, we've worked with customers to enable Lockless code deploys on page 781 over previous releases. This feature has been stable since the last bug fix in version 2019.8.6, and we're confident lockless code deploys are ready for prime time.

Run plans without blocking code deployments
You can allow orchestrator to run plans without blocking your code deployments, and you can deploy code without waiting for plans to finish. For instructions, refer to Running plans alongside code deployments on page 669.

Important: When you enable this feature, Puppet functions or plans that call other plans might behave unexpectedly if a code deployment occurs while the plan is running.

Enhancements

Query jobs by timestamp minimums and maximums
The /jobs and /plan_jobs endpoints have two new query parameters:

- min_finish_timestamp returns only the jobs finished at or after the given UTC timestamp.
- max_finish_timestamp returns only the jobs finished at or before the given UTC timestamp.

File sync client time out
When polling for new commits, if the file sync client doesn't receive data from the file sync storage service for 30 seconds, the file sync client times out.

Optimized some PuppetDB queries
Improved performance of queries that puppet infrastructure uses to look up Puppet infrastructure node certnames.

Upgraded Bouncy Castle
We are now shipping Bouncy Castle 1.70, which has improved support for TLS 1.3.

Report compilation failure results for apply blocks
If the compilation for a node targeted fails while running a plan with an apply block, the console now displays error results on the Plan details page. The results are stored in the database and can be queried.

Platform support
This version adds support for these platforms:

Primary server
- FIPS 140-2 compliant Red Hat Enterprise Linux 8 x86_64
- SUSE Linux Enterprise Server 15 x86_64
- Ubuntu (General Availability kernels) 20.04 amd64

Agent
- macOS 12 x86_64
Resolved issues

Code Manager didn’t delete spec directories
Code Manager deploys failed to delete spec directories due to an issue with r10k. To fix this, we upgraded r10k to version 3.14.1.

LDAP connection timeout setting was ignored
In the console, the LDAP connect_timeout field wasn’t correctly passed to the connection attempt and, as a result, the value was not applied in the LDAP configuration.

The plan that upgrades the secondary node sometimes failed
When running puppet infrastructure upgrade compiler or puppet infrastructure upgrade replica, the plan that upgrades the secondary node sometimes failed on the step where install.bash runs on the node to update the agent.

Failed code compilations in apply blocks didn’t report as failed
When a code compilation failed in a plan apply block, the nodes targeted in the apply block continued to report as in progress rather than reporting as failed.

Subsequent code deployments didn’t report duplicate errors
In previous PE versions, if a code deployment encountered an error on the initial deployment, and the same error occurred on a subsequent deployment, the subsequent deployment incorrectly reported a successful deployment.
Now, subsequent code deployments consistently report errors that were originally encountered on earlier deployments (if the error is still present during the subsequent deployment).

PE 2019.8.10
Released February 2022

Enhancements

Bypass node verification failure when primary server certificates are damaged
You can use --force to bypass node verification failure and force certificate regeneration when your primary server certificates are damaged.

Simplified login error message
The Puppet Enterprise (PE) console login error message instructs the user to contact an administrator rather than consult the console-services log.

Improved disk usage when syncing certificate authority data
Disk usage is better when syncing certificate authority data between the primary and replica.

Platform support
This version adds support for these platforms.

Agent
• Microsoft Windows Server 2022 x86_64
• Red Hat Enterprise Linux 9 x86_64

Patch management
• Amazon Linux 2
Deprecations and removals

Platforms deprecated
Support for these platforms is deprecated in this release.

Primary server
• CentOS 8

Agent
• CentOS 8

Resolved issues

Expired GPG key caused install failures
The GPG key bundled with PE versions prior to 2019.8.4 expired on 17 August 2021, which could cause a failure when PE packages are being added to the system. Customers upgrading to a newer version are no longer affected.

Packages were not marked as automatically installed by APT if you set security_only to true
When running the `pe_patch::patch_server` task with the `security_only` parameter set to true, packages were not marked as being automatically installed by APT. This caused problems if you rely on this marking for APT autoremove to remove old packages. The `patch_server` task now marks packages as automatically installed, regardless of the `security_only` parameter.

r10k can recover from typos in the config or Puppetfile
r10k now updates its cache repos when the remote URL changes. This allows r10k to recover from typos in the config or Puppetfile.

Couldn't complete restoration if the r10k_remote parameter wasn't set
The `r10k_remote` parameter wasn't set when restoring a backup with `scope` set to `certs, code, or puppetdb`. This prevented you from finishing restoration because the commands necessary to finish restoring your primary server did not print.

The fail_plan function didn't show custom error information
The `fail_plan` function ignored the `kind` and `details` parameters, which are used to specify custom, machine-parseable information about an error.

Scheduled jobs failed on FIPS installs
Scheduled jobs, including tasks and plans, couldn't run or be listed on FIPS installs of PE and resulted in the error `javax.crypto.BadPaddingException`.

PE 2019.8.9
Released November 2021

Enhancements

TLS v1.3 is enabled by default
PE is now compatible with TLSv1.2 and TLSv1.3 by default for both FIPS and non-FIPS installations. To update your protocol or ciphers, review the Configuring security settings docs. For a list of compatible ciphers, see the Ciphers reference.
Run patches sequentially in the group_patching plan

The `pe_patch::group_patching` plan now has a parameter called `sequential_patching`, which defaults to `false` (disabled). When set to `true`, nodes in the specified patch group are patched, rebooted (if needed), and the post-reboot script run (if specified) one at a time, rather than all at once.

Run the puppet infra run command with WinRM

The command `puppet infra run` now supports a `--use-winrm` flag, which forces the `run` command to connect to nodes via WinRM and use Bolt instead of the orchestrator.

More options when running the support script

This version of PE includes version 3 of the PE support script, which offers more options for modifying the support script's behavior to meet your needs.

full-deploy setting in Code Manager

By default, Code Manager now utilizes the `r10k --incremental` deploys feature for improved performance. Incremental deploys only sync modules whose definitions allow their version to "float" (such as Git branches) and modules whose definitions have been added or changed since the environment's last deployment. SVN modules are not supported.

To disable this behavior (and deploy all module code regardless of change or float status), set Code Manager's `full_deploy` parameter to `true`.

Platform support

This version adds support for these platforms.

**Primary server**
- AlmaLinux x86_64 for Enterprise Linux 8
- Rocky Linux x86_64 for Enterprise Linux 8

**Agent**
- Ubuntu 18.04 aarch64
- Debian 11 (Bullseye) amd64
- Red Hat Enterprise Linux 8 FIPS x86_64
- AlmaLinux x86_64 for Enterprise Linux 8
- Rocky Linux x86_64 for Enterprise Linux 8

Resolved issues

**Client-side lockfiles were not deleted on startup**

Server-side lockfiles were cleaned up on startup by Puppet Server, but client-side lockfiles were not. Now both client- and server-side lockfiles are deleted during the Puppet Server startup process.

**r10k deleted files in environments pointed to by symlinks**

In control repositories containing symlinks, r10k incorrectly interpreted the files in symlinked locations as duplicates and deleted these files.

**Failed or in-progress reboots reported that they finished rebooting successfully**

When rebooting a node using the `pe_patch::group_patching` plan, the check to detect if a node rebooted always detected that it finished rebooting successfully, even if the reboot failed or was still in progress, due to a
parsing error in the output. This behavior was observed and tested on RHEL-based platform versions 6 and 7, and SLES version 12, but might have existed on other platforms as well.

**Windows agent installation failed if user name contained a space**

The Windows agent install script failed if executed with a user name that included a space, like Max Spacey. You received the error *Something went wrong with the installation* along with exit code 1639. You can now use spaces in usernames without causing a failure.

**Configuring environmentdir to be a relative path caused deploy failures**

When deploying modules from a Puppetfile using r10k or Code Manager, the deploy failed if your environmentdir was configured to be a relative path instead of an absolute path (default).

**The puppet code tool output informational data to stderr**

A regression in the puppet code tool caused Code Manager to output information to stderr, whether it was successful or not. This was inconvenient if deploys were done through pipelines that were configured to register failures based on stderr output, because the behavior of puppet code always led to a failure notification. Now, the notification is printed to stdout instead of stderr.

**The puppet plan subcommand segfaulted**

When run without arguments, the puppet plan subcommand segfaulted. A check was added to ensure the command has arguments set when called.

**PE 2019.8.8**

Released September 2021

**Enhancements**

**Code Manager support for Forge authentication**

Code Manager now supports authentication to custom servers. You can configure this authentication via hieradata by setting authorization_token within the forge_settings parameter:

```bash
---
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: "https://private-forge.mysite"
  authorization_token: "Bearer mysupersecretauthtoken"
---
```

You must prepend the token with 'Bearer', particularly if you use Artifactory as your Forge server.

**Puppet metrics collector module included in PE installation**

The Puppet metrics collector module is now included in PE installations and upgrades. The module is disabled by default. To enable the module to collect system metrics, change these parameters to true:

```bash
puppet_enterprise::enable_metrics_collection: true
puppet_enterprise::enable_system_metrics_collection: true
```

If you have already downloaded the module from the Forge, you must either uninstall your copy of the module or upgrade it to the version installed with PE.
**PE databases module included in PE installation**

The `pe_databases` module is now included in PE installations and upgrades. The module is disabled by default, but can be enabled by setting this parameter to `true`:

```
puppet_enterprise::enable_database_maintenance: <true>
```

If you have already downloaded the module from the Forge, we recommend you upgrade to the version installed with PE.

**Query by order and view timestamps in GET /plan_jobs endpoint**

The GET /plan_jobs endpoint response now includes a timestamp field, and you can include the following sorting parameters `order` and `order_by` in your request.

**Faster Code Manager deploys**

Code Manager deploys are now faster because unmanaged resources are more efficiently purged.

**Platform support**

This version adds support for these platforms.

**Client tools**

- macOS 11

**Deprecations and removals**

**Platforms deprecated**

Support for these agent platforms is deprecated in this release.

- Enterprise Linux 5
- Fedora 30, 31
- macOS 10.14

**Resolved issues**

**r10k refactors erroneously passed flag into modules and broke impact analysis**

Recent r10k refactors broke Continuous Delivery for PE's impact analysis in the 2019.8.7 and 2021.2.0 release. These refactorers passed a `default_branch_override` to r10k via Code Manager's API. r10k erroneously passed the flag to all modules created when the Puppetfile was parsed. This flag was not supported for Forge modules and displayed the following error:

```
ERROR  -> Failed to evaluate /etc/puppetlabs/code-staging/environments/
production_cdpe_ia_1624622874129/Puppetfile
Original exception:
  R10K::Module::Forge cannot handle option 'default_branch_override'
```

This bug is now fixed.

**Replica promotion could fail in air-gapped installations**

If your primary server included AIX or Solaris `pe_repo` classes, replica promotion failed in air-gapped environments because the staged AIX and Solaris tarballs weren't copied to the replica.
**r10k deployment purge level was unsafe when run with parallel deploys**

Previously, Code Manager occasionally failed and returned an HTTP 500 error during environment deployments. This error occurred because of how Code Manager handled a bug/race condition when using pools of r10k caches. This bug also affected Continuous Delivery for PE users. Now, Continuous Delivery for PE users no longer encounter issues related to this race condition and Code Manager's parallel deploys no longer conflict each other.

**PE 2019.8.7**

Released June 2021

**Enhancements**

**Update CRLs**

You can now update your CRLs using the new API endpoint: `certificate_revocation_list`. This new endpoint accepts a list of CRL PEMs as a body, inserting updated copies of the applicable CRLs into the trust chain. The CA updates the matching CRLs saved on disk if the submitted ones have a higher CRL number than their counterparts. You can use this endpoint if your CRLs require frequent updates. Do not use the endpoint to update the CRL associated with the Puppet CA signing certificate (only earlier ones in the certificate chain).

**Filter by node state in jobs endpoint**

You can filter the nodes by their current state in the `/jobs/:job-id/nodes` endpoint when retrieving a list of nodes associated with a given job. The following node states are available to query:

- new
- ready
- running
- stopping
- stopped
- finished
- failed

**Sort activities by oldest to newest in events endpoint**

In the activity service API, the `/v1/events` and `/v2/events` endpoints now allow you to sort activities from either oldest to newest (`asc`) or newest to oldest (`desc`).

**Export node data from task runs to CSV**

In the console, on the Task details page, you can now export the node data results from task runs to a CSV file by clicking Export data.

**Disable force-sync mode**

File sync now always overrides the contents of the live directory when syncing. This default override corrects any local changes made in the live directory outside of Code Manager's workflow. You can no longer disable file sync's force-sync mode to implement this enhancement.

**Differentiate backup and restore logs**

Backup and restore log files are now appended with timestamps and aren't overwritten with each backup or restore action. Previously, backup and restore logs were created as singular, statically named files, `backup.log` and `restore.log`, which were overwritten on each execution of the scripts.
Encrypt backups
You can now encrypt backups created with the `puppet-backup create` command by specifying an optional `--gpgkey`.

Clean up old PE versions with smarter defaults
When cleaning up old PE versions with `puppet infrastructure run remove_old_pe_packages`, you no longer need to specify `pe_version=current` to clean up versions prior to the current one. current is now the default.

Platform support
This version adds support for these platforms.

Agent
- macOS 11
- Red Hat Enterprise Linux 8 ppc64le
- Ubuntu 20.04 aarch64
- Fedora 34

Deprecations and removals

Replace `purge-whitelist` with `purge-allowlist`
For Code Manager and file sync, the term `purge-whitelist` is deprecated and replaced with the new setting name `purge-allowlist`. The functionality and purpose of both setting names are identical.

Platforms deprecated
Support for these agent platforms is deprecated in this release.
- Debian 8

Resolved issues

Windows agent installation failed with a manually transferred certificate
Performing a secure installation on Windows nodes by manually transferring the primary server CA certificate failed with the connection error: `Could not establish trust relationship for the SSL/TLS secure channel`.

Upgrading a replica failed after regenerating the master certificate
If you previously regenerated the certificate for your master, upgrading a replica from 2019.6 or earlier could fail due to permission issues with backed up directories.

The `apply` shim in pxp-agent didn't pick up changes
When upgrading agents, the `ruby_apply_shim` didn't update properly, which caused plans containing `apply` or `apply_prep` actions to fail when run through the orchestrator, and resulted in this error message:

```
Exited 1:
/root/puppetlabs/pzp-agent/tasks-cache/apply_ruby_shim/
apply_ruby_shim.rb:39:in `<main>`: undefined method `map` for nil:NilClass
(NoMethodError)
```
Running client tool commands against a replica could produce errors

Running `puppet-code`, `puppet-access`, or `puppet query` against a replica produced an error if the replica certificate used the legacy common name field instead of the subject alt name. The error has been downgraded to a warning, which you can bypass with some minimal security risk using the flag `--use-cn-verification` or `-k`. To permanently fix the issue, you must regenerate the replica certificate: `puppet infrastructure run regenerate_replica_certificate target=<REPLICA_HOSTNAME>`.

Generating a token using `puppet-access` on Windows resulted in zero-byte token file error

Running `puppet-access login` to generate a token on Windows resulted in a zero-byte token file error. This is now fixed due to the token file method being changed from `os.chmod` to `file.chmod`.

Invoking `puppet-access` when it wasn't configured resulted in unhelpful error

If you invoked `puppet-access` while it was missing a configuration file, it failed and returned unhelpful errors. Now, a useful message displays when `puppet-access` needs to be configured or if there is an unexpected answer from the server.

Enabling `manage_delta_rpm` caused agent run failures on CentOS and RHEL 8

Enabling the `manage_delta_rpm` parameter in the `pe_patch` class caused agent run failures on CentOS and RHEL 8 due to a package name change. The `manage_delta_rpm` parameter now appropriately installs the `drpm` package, resolving the agent run issue.

Editing a hash in configuration data caused parts of the hash to disappear

If you edited configuring data with hash values in the console, the parts of the hash that did not get edited disappeared after committing changes—and then reappeared when the hash was edited again.

Null characters in task output caused errors

Tasks that print null bytes caused an orchestrator database error that prevented the result from being stored. This issue occurred most frequently for tasks on Windows that print output in UTF-16 rather than UTF-8.

Plans still ran after failure

When pe-orchestration-services exited unexpectedly, plan jobs sometimes continued running even though they failed. Now, jobs are correctly transitioned to failed status when pe-orchestration-services starts up again.

Plan apply activity logging contained malformed descriptions

In activity entries for plan apply actions, the description was incorrectly prepended with `desc`.

Patch task failed on Windows nodes with old logs

When patching Windows nodes, if an existing patching log file was 30 or more days old, the task failed trying to both write to and clean up the log file.

Errors when enabling and disabling versioned deploys

Previously, if you switched back and forth from enabling and disabling versioned deploys mode, file sync failed to correctly manage deleted control repository branches. This bug is now fixed.
Lockless code deployment lead to failed removal of old code directories

Previously, turning on lockless code deployment led to full disk utilization because of the failed removal of previous old code directories. To work around this issue, you must manually delete existing old directories. However, going forward—the removal is automatic.

PE 2019.8.6

Released May 2021

Enhancements

Customize value report estimates

You can now customize the low, med, and high time-freed estimates provided by the PE value report by specifying any of the value_report_* parameters in the PE Console node group in the puppet_enterprise::profile::console class.

Re-download CRL on a regular interval

You can now configure the new parameter crl_refresh_interval to re-download the agent's CRL on a regular interval. Use the console to configure the interval in the PE Agent group, in the puppet_enterprise::profile::agent class, and enter a duration (e.g. 60m) for Value.

Remove staging directory status for memory, disk usage, and timeout error improvements

The status output of the file sync storage service (specifically at the debug level), no longer reports the staging directory’s status. This staging information removal reduces timeout errors in the logs, removes heavy disk usage created by the endpoint, and preserves memory if there are many long-running status checks in Puppet Server.

Exclude events from usage endpoint response

In the /usage endpoint, the new events parameter allows you to specify whether to include or exclude event activity information from the response. If set to exclude, the endpoint only returns information about node counts.

Return sensitive data from tasks

You can now return sensitive data from tasks by using the _sensitive key in the output. The orchestrator then redacts the key value so that it isn't printed to the console or stored in the database, and plans must include unwrap() to get the value. This feature is not supported when using the PCP transport in Bolt.

Avoid spam during patching

The patching task and plan now log fact generation, rather than echoing Uploading facts. This change reduces spam from servers with a large amount of facts.

Parameter name updates

As part of the ongoing effort to remove harmful terminology, the parameter master_uris was renamed primary_uris.

Platform support

This version adds support for these platforms.

Agent

• Fedora 32
Resolved issues

Upgrade failed with cryptic errors if `agent_version` was configured for your infrastructure `pe_repo` class
If you configured the `agent_version` parameter for the `pe_repo` class that matches your infrastructure nodes, upgrade could fail with a timeout error when the installer attempted to download a non-default agent version. The installer now warns you to remove the `agent_version` parameter if applicable.

Upgrade with versioned deploys caused Puppet Server crash
If `versioned_deploys` was enabled when upgrading to version 2019.8.6 or 2021.1, then the Puppet Server crashed.

Compiler upgrade failed with client certnames defined
Existing settings for client certnames could cause upgrade to fail on compilers, typically with the error `Value does not match schema: {:client-certnames disallowed-key}`.

Compiler upgrade failed with no-op configured
Upgrade failed on compilers running in no-op mode. Upgrade now proceeds on infrastructure nodes regardless of their no-op configuration.

Installing Windows agents with the `.msi` package failed with a non-default `INSTALLDIR`
When installing Windows agents with the `.msi` package, if you specified a non-default installation directory, agent files were nonetheless installed at the default location, and the installation command failed when attempting to locate files in the specified `INSTALLDIR`.

Installing agents failed with GPG key error on select platforms
When installing Puppet agent version 6.21.1 on Enterprise Linux 5 and SUSE Linux Enterprise Server 11 using the installer script, installation failed with an error about a bad GPG key.

Backup failed with an error about the `stockpile` directory
The `puppet-backup create` command failed under certain conditions with an error that the `/opt/puppetlabs/server/data/puppetdb/stockpile` directory was inaccessible. That directory is now excluded from backup.

Patching failed on Windows nodes with non-default agent location
On Windows nodes, if the Puppet agent was installed to a location other than the default C: drive, the patching task or plan failed with the error `No such file or directory`.

Patching failed on Windows nodes when run during a fact generation
The patching task and plan failed on Windows nodes if run during fact generation. Patching and fact generation processes, which share a lock file, now wait for each other to finish before proceeding.

File sync failed to terminate `pe-puppetserver` Java process
The file sync client failed to terminate the `pe-puppetserver` Java process when it shut down because of a sync error.
File sync failed to copy symlinks if versioned deploys was enabled

If you enabled versioned deploys, then the file sync client failed to copy symlinks and incorrectly copied the symlinks' targets instead. This copy failure crashed the Puppet Server.

Injection attack vulnerability in csv exports

There was a vulnerability in the console where .csv files could contain malicious user input when exported. The values =, +, -, and @ are now prohibited at the beginning of cells to prevent an injection attack.

The License page in the console timed out

Some large queries run by the License page caused the page to have trouble loading and timeout.

PE 2019.8.5

Released February 2021

Enhancements

Install the Puppet agent despite issues in other YUM repositories

When installing the Puppet agent on a node, the installer's YUM operations are now limited to the PE repository, allowing the agent to be installed successfully even if other YUM repositories have issues.

Clean up old packages after upgrade

A new command, puppet infrastructure run remove_old_pe_packages pe_version=current cleans up old PE packages remaining at /opt/puppet/packages/public. For pe_version, you can specify a SHA, a version number, or current. All packages older than the specified version are removed.

Get better insight into replica sync status after upgrade

Improved error handling for replica upgrades now results in a warning instead of an error if re-syncing PuppetDB between the primary and replica nodes takes longer than 15 minutes.

Fix replica enablement issues

When provisioning and enabling a replica (puppet infra provision replica --enable), the command now times out if there are issues syncing PuppetDB, and provides instructions for fixing any issues and separately provisioning the replica.

Patch nodes with built-in health checks

The new group_patching plan patches nodes with pre- and post-patching health checks. The plan verifies that Puppet is configured and running correctly on target nodes, patches the nodes, waits for any reboots, and then runs Puppet on the nodes to verify that they're still operational.

Run a command after patching nodes

A new parameter in the pe_patch class, post_patching_scriptpath enables you to run an executable script or binary on a target node after patching is complete. Additionally, the pre_patching_command parameter has been renamed pre_patching_scriptpath to more clearly indicate that you must provide the file path to a script, rather than an actual command.
Patch nodes despite certain read-only directory permissions

Patching files have moved to more established directories that are less likely to be read-only: /opt/puppetlabs/pe_patch for *nix, and C:\ProgramData\PuppetLabs\pe_patch for Windows. Previously, patching files were located at /var/cache/pe_patch and /usr/local/bin for *nix and C:\ProgramData\pe_patch for Windows.

If you use patch-management, keep these implications in mind as you upgrade to this version:

• Before upgrading, optionally back up existing patching log files, located on patch-managed nodes at /var/cache/pe_patch/run_history or C:\ProgramData\pe_patch. Existing log files are deleted when the patching directory is moved.
• After upgrading, you must run Puppet on patch-managed nodes before running the patching task again, or the task fails.

Use Hiera lookups outside of apply blocks in plans

You look up static Hiera data in plans outside of apply blocks by adding the plan_hierarchy key to your Hiera configuration.

See the duration of Puppet and plan runs

New duration, created_timestamp, and finished_timestamp keys allow you to see the duration of Puppet and plan runs in the GET /jobs and GET /plan_jobs on page 743 endpoints.

View the error location in plan error details

Plan functions provide the file and line number where the error occurred in the details key of the error response.

Run plans on PuppetDB queries and node classifier group targets

The params key in the POST /command/environment_plan_run on page 713 endpoint allows you to specify PuppetDB queries and node groups as targets during a plan run.

Use masked inputs for sensitive parameters

The console now uses password inputs for sensitive parameter in tasks and plans to mitigate a potential "over the shoulder" attack vector.

Configure how many times the orchestrator allows status request timeouts

Configure the new allowed_pcp_status_requests parameter to define how many times an orchestrator job allows status requests to time out before the job fails. The parameter defaults to "35" timeouts. You can use the console to configure it in the PE Orchestrator group, in the puppet_enterprise::profile::orchestrator class.

Accept and store arbitrary data related to a job

An optional userdata key allows you to supply arbitrary key-value data to a task, plan, or Puppet run. The key was added to the following endpoints:

• POST /command/deploy
• POST /command/task
• POST /command/plan_run
• POST /command/environment_plan_run

The key is returned in the following endpoints:

• GET /jobs
• GET /jobs/:job-id
- GET /plan_jobs
- GET /plan_jobs:/job-id

**Sort and reorder nodes in node lists**

New optional parameters are available in the GET /jobs/:job-id/nodes endpoint that allow you to sort and reorder node names in the node list from a job.

**Name versioned directories with SHAs**

The file sync client uses SHAs corresponding to the branches of the control repository to name versioned directories. You must deploy an environment to update the directory names.

**Configure failed deployments to display r10k stacktrace in error output**

Configure the new r10k_trace parameter to include the r10k stack trace in the error output of failed deployments. The parameter defaults to false. Use the console to configure the parameter in the PE Master group, in the puppet_enterprise::master::code_manager class, and enter true for Value.

**Reduce query time when querying nodes with a fact filter**

When you run a query in the console that populates information on the Status page to PuppetDB, the query uses the optimize_drop_unused_joins feature in PuppetDB to increase performance when filtering on facts. You can disable drop-joins by setting the environment variable PE_CONSOLE_DISABLE_DROP_JOINS=yes in /etc/sysconfig/pe-console-services and restarting the console service.

**Resolved issues**

**PuppetDB restarted continually after upgrade with deprecated parameters**

After upgrade, if the deprecated parameters facts_blacklist or cert_whitelist_path remained, PuppetDB restarted after each Puppet run.

**Tasks failed when specifying both as the input method**

In task metadata, using both for the input method caused the task run to fail.

**Patch task misreported success when it timed out on Windows nodes**

If the pe_patch::patch_server task took longer than the timeout setting to apply patches on a Windows node, the debug output noted the timeout, but the task erroneously reported that it completed successfully. Now, the task fails with an error noting that the task timed out. Any updates in progress continue until they finish, but remaining patches aren't installed.

**Orchestrator created an extra JRuby pool**

During startup, the orchestrator created two JRuby pools - one for scheduled jobs and one for everything else. This is because the JRuby pool was not yet available in the configuration passed to the post-migration-fa function, which created its own JRuby pool in response. These JRuby pools accumulated over time because the stop function didn't know about them.

**Console install script installed non-FIPS agents on FIPS Windows nodes**

The command provided in the console to install Windows nodes installed a non-FIPS agent regardless of the node's FIPS status.
Unfinished sync reported as finished when clients shared the same identifier

Because the orchestrator and puppetserver file-sync clients shared the same identifier, Code Manager reported an unfinished sync as "all-synced": true. Whichever client finished polling first, notified the storage service that the sync was complete, regardless of the other client's sync status. This reported sync might have caused attempts to access tasks and plans before the newly-deployed code was available.

Refused connection in orchestrator startup caused PuppetDB migration failure

A condition on startup failed to delete stale scheduled jobs and prevented the orchestrator service from starting.

Upgrade failed with Hiera data based on certificate extensions

If your Hiera hierarchy contained levels based off certificate extensions, like {{trusted.extensions.pp_role}}, upgrade could fail if that Hiera entry was vital to running services, such as {{java_args}}. The failure was due to the puppet infrastructure recover_configuration command, which runs during upgrade, failing to recognize the hierarchy level.

File sync issued an alert when a repository had no commits

When a repository had no commits, the file-sync status recognized this repository’s state as invalid and issued an alert. A repository without any commits is still a valid state, and the service is fully functional even when there are no commits.

Upgrade failed with infrastructure nodes classified based on trusted facts

If your infrastructure nodes were classified into an environment based on a trusted fact, the recover configuration command used during upgrade could choose an incorrect environment when gathering data about infrastructure nodes, causing upgrade to fail.

Backups failed if a Puppet run was in progress

The puppet-backup command failed if a Puppet run was in progress.

Default branch override did not deploy from the module's default branch

A default branch override did not deploy from the module’s default branch if the branch override specified by Impact Analysis did not exist.

Module-only environment updates did not deploy in Versioned Deploys

Module-only environment updates did not deploy if you tracked a module's branch and redeployed the same control repository SHA, which pulled in new versions of the modules.

PE 2019.8.4

Released November 2020

This version updates the PostgreSQL version to address critical security vulnerabilities.

PE 2019.8.3

Released November 2020
New features

Value report
A new Value report page in the Admin section of the console estimates the amount of time reclaimed by using PE automation. The report is configurable based on your environment. See Value report on page 403 for more information.

Enhancements

Spend less time waiting on puppet infrastructure commands
The puppet infrastructure commands that use plans, for example for upgrading, provisioning compilers, and regenerating certificates, are now noticeably faster due to improvements in how target nodes are verified.

Provision a replica without manually pinning the target node
You're no longer required to manually pin the target replica node to the PE Infrastructure Agent group before running puppet infrastructure provision replica. This action — which ensures that the correct catalog and PXP settings are applied to the replica node in load balanced installations — is now handled automatically by the command.

Configure environment caching
Using new environment timeout settings, you can improve Puppet Server performance by caching long-lived environments and purging short-lived environments. For example, in the PE Master node group, in the puppet_enterprise::master class, set environment_timeout_mode = from_last_used and environment_timeout = 30m to clear short-lived environments 30 minutes from when they were last used. By default, when you enable Code Manager, environment_timeout is set to unlimited, which caches all environments.

Configure the number of threads used to download modules
A new configuration parameter, download_pool_size, lets you specify the number of threads r10k uses to download modules. The default is 4, which improves deploy performance in most environments.

Configure PE-PostgreSQL autovacuum cost limit
The cost limit value used in PE-PostgreSQL autovacuum operations is now set at a more reasonable default that scales with the number of CPUs and autovacuum workers available. The setting is also now configurable using the puppet_enterprise::profile::database::autovacuum_vacuum_cost_limit parameter.

Previously, the setting was not configurable, and it used the PostgreSQL default, which could result in database tables and indexes growing continuously.

Rerun Puppet or tasks on failed nodes only
You can choose to rerun Puppet or a task only on the nodes that failed during the initial run by selecting Failed nodes on the Run again drop down.

Run plans only when required parameters are supplied
In the console, the option to commit a plan appears after you supply all required parameters. This is to prevent plan failures by accidentally running plans without required parameters.
Schedule plans in the console and API
You can use the console to schedule one time or recurring plans and view currently scheduled plans. Additionally, you can use the `schedule_plan` command for scheduling one-time plan runs using the `POST /command/schedule_plan` on page 711 endpoint.

Use sensitive parameters in plans
You can use the Sensitive type for parameters in plans. Parameters marked as Sensitive aren't stored in the orchestrator's database, aren't returned via API calls, and don't appear in the orchestrator's log. Sensitive parameters are also not visible from the console.

View more details about plans
The environment a plan was run from is now displayed in the console on the Job details page. The environment is returned from the `/plan_jobs` endpoint using the new `environment` key. See GET `/plan_jobs` on page 743 for more information.

Additionally, the parameters supplied to tasks that are run as part of a plan are displayed in the console on the Plan details page. Sensitive parameters are masked and are never stored for a task run.

Differentiate between software and driver patch types for Windows
PE now ignores Driver update types in Windows Updates by default and only includes the Software type, cutting down on unnecessary patch updates. To change this default, configure the new `windows_update_criteria` parameter in the `pe_patch` class by removing or changing the Type argument. See Patch management parameters on page 596 for more information about the parameter.

Serve patching module files more efficiently
Certain `pe_patch` module files are now delivered to target nodes using methods that improve scalability and result in fewer file metadata checks during Puppet runs.

Receive a notification when CA certs are about to expire
The console now notifies you if a CA certificate is expiring soon. The Certificates page in the sidebar displays a yellow ! badge if a certificate expires in less than 60 days, and a red ! badge if a certificate expires in less than 30 days. If there are certificates that need signing in addition to the certificates expiring, the number of certificates that need to be signed is displayed in the badge but the color stays the same.

View details about a particular code deployment
The Code Manager /deploys/status endpoint now includes the deployment ID in the "deploys-status" section for incomplete deploys so you can correlate status output to a particular deployment request.

Additionally, you can query the Code Manager /deploys/status endpoint with a deployment ID to see details about a particular deployment. The response contains information about both the Code Manager deploy and the sync to compilers for the resulting commit.

Troubleshoot code deployments
File sync now uses the public git SHA recorded in the signature field of the .r10k-deploy.json file instead of an internal SHA that file sync created. Additionally, versioned directories used for lockless deploys now use an underscore instead of a hyphen so that paths are valid environment names. With these changes, you can now map versioned directories directly to SHAs in your control repository.

When upgrading to 2019.3 or later with versioned deploys enabled, versioned directories are recreated with underscores. You can safely remove orphaned directories with hyphens located at /opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs.
Report on user activities
A new GET /v2/events on page 389 API that tracks more user activities, like date, time, remote IP address, user ID, and action. You can use the console to generate a report of activities on the User details page.

Platform support
This version adds support for these platforms.

Agent
- Red Hat Enterprise Linux 8 aarch64

Deprecations and removals

Master removed from docs
Documentation for this release replaces the term master with primary server. This change is part of a company-wide effort to remove harmful terminology from our products.

For the immediate future, you’ll continue to encounter master within the product, for example in parameters, commands, and preconfigured node groups. Where documentation references these codified product elements, we’ve left the term as-is.

As a result of this update, if you’ve bookmarked or linked to specific sections of a docs page that include master in the URL, you’ll need to update your link.

Whitelist and blacklist deprecated
In the interest of removing racially insensitive terminology, the terms whitelist and blacklist are deprecated in favor of allowlist and blocklist. Classes, parameters, and file names that use these terms continue to work, but we recommend updating your classification, Hiera data, and pe.conf files as soon as possible in preparation for their removal in a future release.

These are the classes, parameters, task parameters, and file names that are affected.

Classes
- puppet_enterprise::pg::cert_whitelist_entry
- puppet_enterprise::certs::puppetdb_whitelist
- puppet_enterprise::certs::whitelist_entry

Parameters
- puppet_enterprise::master::code_manager::purge_whitelist
- puppet_enterprise::master::file_sync::whitelisted_certnames
- puppet_enterprise::orchestrator::ruby_service::whitelist
- puppet_enterprise::profile::ace_server::whitelist
- puppet_enterprise::profile::bolt_server::whitelist
- puppet_enterprise::profile::certificate_authority::client_whitelist
- puppet_enterprise::profile::console::cache::cache_whitelist
- puppet_enterprise::profile::console::whitelisted_certnames
- puppet_enterprise::profile::puppetdb::sync_whitelist
- puppet_enterprise::profile::puppetdb::whitelisted_certnames
- puppet_enterprise::puppetdb::cert_whitelist_path
- puppet_enterprise::puppetdb::database_ini::facts_blacklist
- puppet_enterprise::puppetdb::database_ini::facts_blacklist_type
- puppet_enterprise::puppetdb::jetty_ini::cert_whitelist_path

Files

© 2022 Puppet, Inc., a Perforce company
Split-to-mono migration removed

The `puppet infrastructure run migrate_split_to_mono` command has been removed. The command migrated a split installation to a standard installation with the console and PuppetDB on the primary server. Upgrades to PE 2019.2 and later required migrating as a prerequisite to upgrade, so this command is no longer used. If you're upgrading from an earlier version of PE with a split installation, see *Migrate from a split to a standard installation* in the documentation for your current version.

Resolved issues

**Upgrade and puppet infrastructure commands failed if your primary server was not in the production environment**

Upgrades and puppet infrastructure commands — including replica upgrade and compiler provisioning, conversion, and upgrade — failed with a `Bolt::RunFailure` if your primary server was not in the production environment.

This release fixes both issues, and upgrades to this version are unaffected.

For upgrades to previous versions, we recommended following these steps to ensure upgrades and puppet infrastructure commands worked as expected:

- Verify that you've specified your non-production infrastructure environment for these parameters:
  - `pe_install::install::classification::pe_node_group_environment`
  - `puppet_enterprise::master::recover_configuration::pe_environment`
  - Run `puppet infra recover_configuration --pe-environment <PRIMARY_ENVIRONMENT>`
  - When upgrading, run the installer with the `--pe_environment` flag: `sudo ./puppet-enterprise-installer --pe_environment <PRIMARY_ENVIRONMENT>`

**Upgrade failed if a PostgreSQL repack was in progress**

If a PostgreSQL repack operation was in progress when you attempted to upgrade PE, the upgrade could fail with the error `cannot drop extension pg_repack because other objects depend on it`.

**Upgrade failed with an unenabled replica**

PE upgrade failed if you had a provisioned, but not enabled, replica.

**Compiler provisioning failed if a single compiler was unresponsive**

The `puppet infrastructure provision compiler` command failed if any compiler in your pool failed a pre-provisioning health check.

**puppet infrastructure commands failed with an external node classifier**

With an external node classifier, puppet infrastructure commands, such as `puppet infrastructure compiler upgrade` and `puppet infrastructure provision compiler`, failed.

**Automated Puppet runs could fail after running compiler or certificate regeneration commands**

After provisioning compilers, converting compilers, or regenerating certificates with puppet infrastructure commands, automated Puppet runs could fail because the Puppet service hadn't restarted.

© 2022 Puppet, Inc., a Perforce company
puppet infrastructure recover_configuration misreported success if specified environment didn’t exist

If you specified an invalid environment when running puppet infrastructure recover_configuration, the system erroneously reported that the environment’s configuration was saved.

Runs, plans, and tasks failed after promoting a replica

After promoting a replica, infrastructure nodes couldn't connect to the newly promoted primary server because the master_uris value still pointed to the old primary server.

This release fixes the issue for newly-provisioned replicas, however if you have an enabled replica, in both the PE Agent and PE Infrastructure Agent node groups, in the puppet_enterprise::profile::agent class, verify that the setting for master_uris matches the setting for server_list. Both values must include both your primary server and replica, for example ["PRIMARY.EXAMPLE.COM", "REPLICA.EXAMPLE.COM"]. Setting these values ensures that agents can continue to communicate with the promoted replica in the event of a failover.

Skipping agent configuration when enabling a replica deleted settings for the PE Agent group

If you used the --skip-agent-config flag with puppet infra enable replica or puppet infra provision replica --enable, any custom settings that you specified for server_list and pcp_broker_list in the PE Agent node group were deleted.

Replica commands could leave the Puppet service disabled

The reinitialize replica command as well as the provision replica command, which includes reinitializing, left the Puppet service disabled on the replica.

Provisioning a replica failed after regenerating the primary server certificate

If you previously regenerated the certificate for your primary server, provisioning a replica can failed due to permission issues with backed up directories.

Console was inaccessible with PE set to IPv6

If you specified IPv6, PE Java services still listened to the IPv4 localhost. This mismatch could prevent access to the console as Nginx proxied traffic to the wrong localhost.

Apply blocks failed to compile

Puppet Server might have failed to compile apply blocks for plans when there were more than eight variables, or when variables had names that conflict with key names for hashes or target settings in plans.

Yaml plans displayed all parameters as optional in the console

Yaml plans listed all parameters as having default values, regardless of whether there is a default value set in the code or not. This caused all parameters to display defaults in orchestrator APIs and show as optional in the console. Yaml plans no longer display all parameters as optional.

Running puppet query produced a cryptic error

Running puppet query with insufficient permissions produced an error similar to this:

```
ERROR - &{<nil> } (*models.Error) is not supported by the TextConsumer, can be resolved by supporting TextUnmarshaller interface
```
Primary server reported HTTP error after Qualys scan
When running a Qualys scan, the primary server no longer reports the error "HTTP Security Header Not Detected. Issue at Port 443".

Nodes CSV export failed with PQL query
The csv export functionality no longer produces an error when you specified nodes using a PQL query.

The `wait_until_available` function didn’t work with multiple transports
When a target included in the `TargetSpec` argument to the `wait_until_available` plan function used the ACE (remote) transport, the function failed immediately and wouldn't wait for any of the targets in the `TargetSpec` argument.

Unnecessary logs and failed connections in `bolt-server` and `ace-server`
When requests were made with unsupported ciphers, `bolt-server` and `ace-server` would log stack traces. Stack traces might lead to unusual growth in the logs for those services when, for example, they are scanned by security scanning products. The Puma Server library in those services has been updated to prevent emitting the stack traces into the `bolt-server.log` and `ace-server.log`.

Patch task could misreport success for Windows nodes
When patching Windows nodes, running the `pe_patch::patch_server` task always reported success, even if there were problems installing one or more updates. With this fix, the task now fails with an error message about which updates couldn't be installed successfully.

The `pe_patch` fact didn’t consider classifier environment group
When `pe_patch` scheduled scripts that uploaded facts, the facts didn't consider the current environment the node was compiling catalogs in. If the local agent environment didn’t match the environment specified by the server, the facts endpoint included tags for an unexpected environment.

Reenabling lockless code deploys could fail
Reenabling lockless code deploys could fail due to the persistence of the versioned code directory. With this release, any existing versioned code directory is deleted – and recreated – when you reenable lockless code deploys.

File-sync client repo grew after frequent commits
The file-sync client repo no longer grows rapidly when there are frequent commits to it. For example, when syncing the CA dir for DR, and many new certificates are signed are revoked quickly.

PE 2019.8.1
Released August 2020

Enhancements

Value reporting
A new values API reports details about automated changes that PE makes to nodes, and provides an estimate of time freed by each type of change based on intelligent defaults or values you provide. You can also specify an average hourly salary and see an estimate of cost savings for all automated changes.

Console navigation and workflow improvements
There are several improvements to the console, including new sections and renamed pages.

© 2022 Puppet, Inc., a Perforce company
• The **Classification** page was renamed **Node groups**.
• The setup page was renamed **Admin**.
• There is a new **Inventory** section in the sidebar, which contains the **Nodes, Node groups, and Packages** pages.
• The **Inventory** page was removed. To add nodes to inventory, click **Add nodes** in the upper right corner of the **Nodes** page.
• There is a new **Access control** page, which contains tabs for **Users, User roles, User groups, and External directory**.
• The **Configuration** tab was broken out into two tabs: **Classes** and **Configuration data**. The **Classes** tab is for declaring classes and setting parameters while the **Configuration data** tab is for setting parameters without declaring classes.
• There is a **New in 2019.8** page in the sidenav, which lists console-related release notes. It will be updated after each z release and is visible for the first two weeks after a release.

**Compiler conversion runs in parallel**

When you convert all compilers at one time with `puppet infrastructure run convert_legacy_compiler all=true`, the process is now noticeably faster due to streamlining in when Puppet runs occur on target hosts.

**Console displays enum and boolean plan parameter values in select menu**

You can select plan parameters that are boolean or enum types from a drop down menu in the **Value** field.

**Updates to metrics endpoints**

Access to endpoints under /metrics are now controlled by trapperkeeper-authorization and configured in the Puppet Server auth.conf file. The default rule allows remote access with a valid Puppet certificate.

Setting the v2 metrics endpoint to debug no longer displays debug messages from Jolokia. In order to see debugging messages, set a configuration value in addition to the usual logback changes.

**Deprecations and removals**

**Application orchestration features in the Puppet language**

Support for application orchestration was dropped in PE version 2019.1 and later. These application orchestration features of the Puppet language are also now deprecated:

• **Keywords**: site, application, consumes, and produces
• **Metaparameters**: export and consume
• **Resource kinds**: application, site, capability_mapping
• **Puppet::Parser::EnvironmentCompiler**
• **Puppet::Parser::Compiler::CatalogValidator::SiteValidator**
• **Puppet::Parser::Compiler::CatalogValidator::EnvironmentRelationshipValidator**
• **Puppet::Type#is_capability?**
• **Puppet::Type#application?**
• **Environment catalog REST API**

**SUSE Linux Enterprise Server dependencies**

SUSE Linux Enterprise Server nodes no longer have a dependency on libboost_* and libyamlcpp packages.
Resolved issues

Upgrading Windows agents using the puppet_agent module could restart non-Puppet services
If you're using a log aggregator, upgrading Windows agents using the puppet_agent module could cause non-Puppet services to restart.

Upgrading agents using the puppet_agent module could produce non-breaking errors
Upgrading agents from versions 6.14 or 6.15 using the puppet-agent module could produce errors about an unavailable file resource or unknown HTTP resource. These errors occurred only during the initial Puppet agent run, when the agent was still using versions 6.14 or 6.15 with an updated primary server. The error resolved after the puppet-agent service restarted.

Pre-upgrade check produced inaccurate errors on standalone PE-PostgreSQL nodes
When you ran the installer on a standalone PE-PostgreSQL node to upgrade it, you saw an error like this:

```bash
## Pre-Upgrade Checks
Warning: Puppet agent is not running.
Error: No configuration file found at /etc/puppetlabs/client-tools/services.conf. This file is installed automatically on Puppet Server nodes.
Make sure you are running the command on a primary master, primary master replica, or compile master.
Error: Try 'puppet infrastructure help status' for usage
```

The error occurred because the pre-upgrade check verified services running on the primary server which were not present on standalone PE-PostgreSQL nodes.

Upgrade could fail with custom structured facts
If you use custom facts that use structured facts, upgrade could fail with an error related to your custom fact, for example: undefined method '[]' for nil:NilClass (Puppet::Error).

Upgrade commands failed if PXP agents were configured to connect to load balancers
In installations with load balancers, the puppet infrastructure upgrade commands could fail if the PXP agent on infrastructure nodes connected to load balancers instead of to the primary server. The upgrade plan now verifies configuration and prompts you to fix any issues before continuing with the upgrade.

Compiler upgrade could fail to upgrade Puppet Server
The puppet infrastructure upgrade compiler command could fail to upgrade Puppet Server depending on how the catalog was built for performing the upgrade.

Converting all legacy compilers failed in disaster recovery installations
With disaster recovery enabled, the command to convert legacy compilers with the option all=true failed.

Converting legacy compilers failed with autosigning enabled
Running puppet infrastructure run convert_legacy_compiler with autosigning enabled caused the conversion to fail during certificate regeneration.
Converting legacy compilers could fail with DNS alternative names

If `dns_alt_names` were specified in the `[agent]` section of `puppet.conf`, the `puppet infrastructure run convert_legacy_compiler` command failed because it didn't recognize the alternative names. As a temporary workaround, we recommended moving `dns_alt_names` to the `[main]` section of `puppet.conf` on the compilers to be converted, however `[agent]` is the preferred section to specify this parameter. The compiler conversion command now recognizes DNS alternative in either the `[agent]` or `[main]` section of `puppet.conf`.

Missing package dependencies for SUSE Linux Enterprise Server agent nodes

On agent nodes running SUSE Linux Enterprise Server 15, the `libyaml-cpp` package and operating system packages prefixed with `libboost_` were no longer bundled with Puppet agent, and also might not have been included in the operating system.

Command to regenerate agent certificates didn't work with nodes behind a load balancer

In large and extra-large installations with load balancers, the command `puppet infrastructure run regenerate_agent_certificate` failed because compilers didn't have the tasks needed to run the command, and agent nodes don't communicate directly with the primary server.

With lockless code deploy enabled, deleted branches could increase disk use

If you deleted a branch from your control repository with lockless deploys enabled, some artifacts could remain on disk and increase your disk use.

With lockless code deploy enabled, deploying with `--wait` could produce an erroneous timeout

Deploying code from the command line or API with the `--wait` flag produced a timeout error, even though the code deploy completed.

The `blackout_windows` parameter in `pe_patch` class couldn't handle time zones with negative UTC offset

If you used a negative value to offset the timezone when setting the `blackout_windows` parameter for patching node groups, the `pe_patch` fact would return an error.

The `pe_patch` fact wouldn't generate if there was a parsing error

The `pe_patch` fact couldn't be generated if there was an error when parsing the latest cached catalog for the node. Additionally, if you did not have `puppetlabs-stdlib` installed, packages that were fixed to a particular version in the node's catalog were not recognized by `pe_patch`.

Node search input didn't respond to `Enter` key

The node name search bar on the `Nodes` page in the console didn't respond to the `Enter` key to search for a node and you had to select `Submit` manually. You can now use `Enter` to search for nodes.

Console radiator bars had a width of zero

In the console, the colored bars in the radiator were broken and didn't show the correct layout. The radiator has been fixed.

PE 2019.8

Released June 2020
New features

Patch management
You can now manage patches on *nix and Windows nodes in the Patch Management section of the console. After setting up patching node groups, you can view the patch status for your nodes, filter available patches by type and operating system, and run a pre-filled task to apply patches to selected nodes from the Patches page. For information on configuring patch management and applying patches, see Managing patches on page 592.

Lockless code deploys
Using Code Manager, you can now optionally deploy code to versioned code directories rather than the live code directory. This change enables Puppet Server to continue serving catalog requests even as you deploy code.

You can enable lockless code deploys by setting puppetenterprise::profile::master::versioned_deploys to true. For more information about lockless code deploys, see Enable lockless code deploys on page 782.

Note: Lockless deploys are an experimental feature which might experience breaking changes in future releases. We recommend testing lockless deploys in a non-production environment before enabling it in production.

Enhancements

Improvements to puppet infrastructure upgrade commands
When you specify more than one compiler to upgrade, the puppet infrastructure upgrade compiler command now upgrades all compilers at the same time, rather than sequentially. Additionally, with both the compiler and replica upgrade commands, you can now specify the location of an authentication token other than the default. For example: puppet infrastructure upgrade compiler --token-file=<PATH_TO_TOKEN>.

More secure code deploys
Permissions for the Puppet code directory are now managed by file sync directly, instead of relying on symlinks. This change improves security during code deployment.

Logging for puppet infrastructure commands that use the orchestrator
A new log file located at /var/log/puppetlabs/installer/orchestrator_info.log contains run details about puppet infrastructure commands that use the orchestrator, including the commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.

Improved error handling for plans
Before running plans, the built-in check for node connectivity now provides more descriptive error messages, such as host key verification failures.

Unspecified default values for tasks and plans are supplied automatically
When you run tasks or plans through the orchestrator API or the console, default values are supplied to the job when the user does not supply them. Parameters that have a default value are treated as optional in the console.

Note: The parameters you supply the first time you run a task will be used for subsequent task runs when using the Run again feature on the Task details page.

New scheduling options in the console
You can now specify scheduled tasks and Puppet jobs to run every two weeks or every four weeks.
Plan support for `apply()` on pcp transports

Plans now support using the `apply_prep()` function and blocks of Puppet code within calls to `apply()`. The feature is only available on targets connected to PE using the PCP transport and does not work on nodes connected over SSH or WinRM.

Support for new options in the command/deploy endpoint

The following options were added to the orchestrator's `POST /command/deploy` on page 630 endpoint:

- `filetimeout`
- `http_connect_timeout`
- `http_keepalive_timeout`
- `http_read_timeout`
- `ordering`
- `skip_tags`
- `tags`
- `use_cached_catalog`
- `usecacheonfailure`

Platform support

This version adds support for these platforms:

Agent

- macOS 10.15

Deprecations and removals

Razor removed

Razor has been removed from PE in this release. If you want to continue using Razor, you can use the open source version of the tool.

Support for `bolt.yaml` settings in plans removed

Settings from `bolt.yaml` are no longer read from the environment directory. The `modulepath` setting is only configurable from `environment.conf`.

Platforms removed

Support for these platforms is removed in this release:

Primary server

- Enterprise Linux 6
- Ubuntu 16.04

Resolved issues

Upgrade removed custom classification rules from PE Master node group

Custom rules that you used to classify compilers in the PE Master node group were removed upon upgrade, or when you ran `puppet infrastructure configure`.

Upgrade failed with a `Could not retrieve facts` error

If an invalid custom fact installed on the primary server couldn't be resolved during upgrade, the upgrade failed with an error similar to this:
Could not retrieve facts ... undefined method `split' for nil:NilClass
(Puppet::Error)from /opt/puppetlabs/installer/lib/ruby/gems/2.5.0/gems/facter-4.0.20/lib/custom_facts/util/loader.rb:125:in `load'

**Upgrading a replica could temporarily lock the agent on the primary server**

If you tried to run Puppet on your primary server before the `puppet infrastructure upgrade replica` command completed, you could encounter an error that a Puppet run was already in progress.

**FIPS installs didn't fully support cert chain validation**

In FIPS environments, RBAC could not connect to LDAP using a pem or jks file.

**Command to remove old PostgreSQL versions failed on Ubuntu**

When run on Ubuntu nodes, the `puppet infrastructure run remove_old_postgresql_versions` command failed, erroneously reporting that PostgreSQL wasn't installed.

**Enabling a replica could fail immediately after provisioning**

When running `puppet infrastructure provision replica --enable`, the command could fail after the replica was provisioned but before it was enabled if services on the replica were still starting up. The command now waits for services to start and verifies that replication has completed before enabling the replica.

**Ubuntu 20.04 couldn't be installed with PE package management**

Ubuntu 20.04 wasn't available for installation as a `pe_repo` class, even though it was a supported agent platform.

**Loading plan lists crashed console services**

When plan run results were large, the console crashed due to high memory usage on the Plan details page. An optional results query parameter has been added to the GET/plan_jobs endpoint. This parameter keeps you from experiencing high memory usage in the console when loading results for large plan runs.

**Default value for tasks and plans dropped in middleware**

When a task had a default value of `false` or `null`, the console metadata panel did not display the default value.

**Event inspector displayed wrong table types**

Browsing the event inspector sometimes created inconsistencies in tables and errors in table links.

## PE known issues

These are the known issues in PE 2019.8.

## Installation and upgrade known issues

These are the known issues for installation and upgrade in this release.

### Expired GPG key causing install failures

**Note:** This issue was fixed in version 2019.8.10. This known issue applies to earlier versions.

The GPG key bundled with PE versions prior to 2019.8.4 expired on 17 August 2021. If you’re installing version 2019.8.9 or earlier of PE on Debian, Ubuntu, or SUSE Linux Enterprise Server (SLES) nodes, the expired key might cause a failure when PE packages are being added to the system.

To fix the issue, do one of the following:
• On Debian or Ubuntu, do the workaround described in **Expired GPG Key causes node installation to fail.**
• On SLES, on the signature verification failure message, select yes when asked if you want to continue with the installation. Here is an example of the message:

```
Signature verification failed for file 'repomd.xml' from repository 'puppet-enterprise'.

Note: Signing data enables the recipient to verify that no modifications occurred after the data were signed. Accepting data with no, wrong or unknown signature can lead to a corrupted system and in extreme cases even to a system compromise.

Note: File 'repomd.xml' is the repositories master index file. It ensures the integrity of the whole repo.

Warning: This file was modified after it has been signed. This may have been a malicious change, so it might not be trustworthy anymore! You should not continue unless you know it’s safe.
```

**Signature verification failed for file 'repomd.xml' from repository 'puppet-enterprise'. Continue? [yes/no] (no): yes**

---

**Initial agent run after upgrade can fail with many environments**

In installations with many environments, where file sync can take several minutes, the orchestration service fails to reload during the first post-upgrade Puppet run. As a workaround, re-run the Puppet agent until the orchestration service loads properly. To prevent encountering this error, you can clean up unused environments before upgrading, and wait several minutes after the installer completes to run the agent.

**Converting legacy compilers fails with an external certificate authority**

If you use an external certificate authority (CA), the puppet infrastructure run `convert_legacy_compiler` command fails with an error during the certificate-signing step.

```
Agent_cert_regen: ERROR: Failed to regenerate agent certificate on node <compiler-node.domain.com>
Agent_cert_regen: bolt/run-failure:Plan aborted: run_task 'enterprise_tasks::sign' failed on 1 target
Agent_cert_regen: puppetlabs.sign/sign-cert-failed Could not sign request for host with certname <compiler-node.domain.com> using caserver <master-host.domain.com>
```

To work around this issue when it appears:

1. Log on to the CA server and manually sign certificates for the compiler.
2. On the compiler, run Puppet: `puppet agent -t`
3. Unpin the compiler from **PE Master** group, either from the console, or from the CLI using the command: /opt/puppetlabs/bin/puppet resource pe_node_group "PE Master" unpinned="<COMPILER_FQDN>"
4. On your primary server, in the `pe.conf` file, remove the entry `puppet_enterprise::profile::database::private_temp_puppetdb_host`
5. If you have an external PE-PostgreSQL node, run Puppet on that node: `puppet agent -t`
6. Run Puppet on your primary server: `puppet agent -t`
7. Run Puppet on all compilers: `puppet agent -t`
Converted compilers can slow PuppetDB in multi-region installations

In configurations that rely on high-latency connections between your primary servers and compilers – for example, in multi-region installations – converted compilers running the PuppetDB service might experience significant slowdowns. If your primary server and compilers are distributed among multiple data centers connected by high-latency links or congested network segments, reach out to Support for guidance before converting legacy compilers.

Ruby gem installation failures

In 2019.8.7, installing ruby gems with native extensions, like digest-crc, can result in failures.

Disaster recovery known issues

These are the known issues for disaster recovery in this release.

An unreachable replica runs the primary server out of disk space

If a provisioned replica becomes unreachable, the associated primary server can quickly run out of disk space, causing a complete interruption to PE services. In larger installations, an outage can occur in under an hour. The disk usage issue is caused by the PE-PostgreSQL service on the primary server retaining change logs that the replica hasn’t acknowledged.

To avoid an outage, resolve any replica issues immediately, or forget the replica: from your primary server, run puppet infrastructure forget <REPLICANODE NAME>.

FIPS known issues

These are the known issues with FIPS-enabled PE in this release.

c_rehash binary vulnerability

While CVE-2022-1292 and CVE-2022-2068 were resolved for non-FIPS installations in PE 2019.8.12, these vulnerabilities are still present for FIPS installations. To mitigate this vulnerability, use this command to make the c_rehash binary non-executable:

```bash
chmod -x /opt/puppetlabs/puppet/bin/c_rehash
```

Puppet Server FIPS installations don’t support Ruby’s OpenSSL module

FIPS-enabled PE installations don’t support extensions or modules that use the standard Ruby Open SSL library, such as hiera-eyaml. As a workaround, you can use a non-FIPS-enabled primary server with FIPS-enabled agents, which limits the issue to situations where only the primary uses the Ruby library. This limitation does not apply to versions 1.1.0 and later of the splunk_hec module, which supports FIPS-enabled servers. The FIPS Mode section of the module's Forge page explains the limitations of running this module in a FIPS environment.

Configuration and maintenance known issues

These are the known issues for configuration and maintenance in this release.

regenerate_agent_certificate can’t verify node type if client tools were installed through a package resource

When running the puppet infra run regenerate_agent_certificate command, the plan can't verify that a node isn't an infrastructure node if the pe-client-tools package was installed on the node through a package resource. To make the plan succeed, you must pass in: force=true
Restarting or running Puppet on infrastructure nodes can trigger an illegal reflective access operation warning

When restarting PE services or performing agent runs on infrastructure nodes, you might see the warning Illegal reflective access operation ... All illegal access operations will be denied in a future release in the command-line output or logs. These warnings are internal to PE service components, have no impact on their functionality, and can be safely disregarded.

Orchestration services known issues

These are the known issues for the orchestration services in this release.

Orchestrator doesn't properly periodically prune jobs

A calculation error introduced in 2019.8.10 causes job records to be stored beyond the job_prune_threshold limit. Restarting pe-orchestration-services triggers pruning.

Running plans during code deployment can result in failures

If a plan is running during a code deployment, things like compiling apply block statements or downloading and running tasks that are part of a plan might fail. This is because plans run on a combination of PE services, like orchestrator and puppetserver, and the code each service is acting on might get temporarily out of sync during a code deployment.

Console and console services known issues

These are the known issues for the console and console services in this release.

Gateway timeout errors in the console

In versions 2019.8.0 through 2019.8.4, using facts to filter nodes might produce either a “502 Bad Gateway” or "Gateway Timeout" error instead of the expected results. This was resolved in 2019.8.5.

Patching known issues

These are the known issues for patching in this release.

Patching fails with excluded YUM packages

In the patching task or plan, using yum_params to pass the --exclude flag in order to exclude certain packages can result in task or plan failure if the only packages requiring updates are excluded. As a workaround, use the versionlock command (which requires installing the yum-plugin-versionlock package) to lock the packages you want to exclude at their current version. Alternatively, you can fix a package at a particular version by specifying the version with a package resource for a manifest that applies to the nodes to be patched.

Code management known issues

These are the known issues for Code Manager, r10k, and file sync in this release.

Can't configure full_deploy parameter

Configuring module deployment scope on page 785 by adding the full_deploy parameter to your Code Manager configuration in Hiera does not correctly modify the parameter's setting.

Until this issue is fixed, you'll need to create a separate config file to manage this parameter. To do this:

1. In the same directory as your code-manager.conf file, create a .conf file with the following contents:

```plaintext
code-manager: {
    full-deploy: true
}
```
2. Set the same permissions on this file as code-manager.conf:

```
chown pe-puppet:pe-puppet /etc/puppetlabs/puppetserver/conf.d/<FILENAME>.conf
chmod 640 /etc/puppetlabs/puppetserver/conf.d/<FILENAME>.conf
```

3. Restart pe-puppetserver:

```
systemctl restart pe-puppetserver
```

**Important:** PE doesn't manage this configuration file. The file can persist through upgrades, so you'll need to remove it once the issue is resolved, and then follow the steps in Configuring module deployment scope on page 785 to reconfigure the parameter.

### Changing a file type in a control repo produces a checkout conflict error

Changing a file type in a control repository – for example, deleting a file and replacing it with a directory of the same name – generates the error `JGitInternalException: Checkout conflict with files accompanied by a stack trace in the Puppet Server log`. As a workaround, deploy the control repo with the original file deleted, and then deploy again with the replacement file or directory.

### Enabling Code Manager and multithreading in Puppet Server deadlocks JRuby

Setting the new `environment_timeout` parameter to any non-zero value – including the `unlimited` default when Code Manager is enabled – interferes with multithreading in Puppet Server and can result in JRuby deadlocking after several hours.

### Code Manager and r10k do not identify the default branch for module repositories

When you use Code Manager or r10k to deploy modules from a Git source, the default branch of the source repository is always assumed to be main. If the module repository uses a default branch that is not main, an error occurs. To work around this issue, specify the default branch with the `ref:` key in your Puppetfile.

### What's new since PE 2018.1

These are the major new features, enhancements, deprecations, and removals since the previous LTS release (2018.y). Most bug fixes were ported back to PE 2018.1 if they affected that version, so there aren't any notable resolved issues between LTS versions.

This list is not intended to be exhaustive, nor to be specific about which interim release a feature or change applied to. For a breakdown of interim releases, see the applicable release notes.

|---------------|--------|--------|--------|---------------|

**Important:** For details about features and fixes that might affect your upgrade, see Upgrade cautions on page 258.

Get familiar with the latest hardware requirements, supported operating systems and browsers, and network configuration details in System requirements on page 182.
New features

- **Simplified installation**
  A simplified PE installation process now relies entirely on defaults, apart from specifying a console administrator password. After installation, you can scale or configure your environment as needed.

  You can still install PE using a customized `pe.conf` file by running `./puppet-enterprise-installer -c /tmp/pe.conf`.

- **FIPS compliance**
  PE is now fully operable on select FIPS platforms.

- **Agent installation from the console**
  You can now add *nix and Windows agents to target nodes from the console using `pe_bootstrap` tasks.

- **Support for agentless nodes and network devices**
  You can now add nodes to your inventory without installing the Puppet agent. When you add nodes and their credentials to the inventory, the information is securely stored and made available in the console through remote connections. Authorized users can then run tasks on these nodes without re-entering credentials.

  You can also add network devices to inventory. This feature requires that you have installed modules for device transports in your production environment. By adding connections to those devices to your inventory, you can manage switches and firewalls, and run Puppet and tasks on them, just like other agentless nodes in your infrastructure.

- **Plans in PE**
  You can now run plans in PE from the console, on the command line, or using the orchestrator API `/command/plan_run` endpoint. A plan is a bundle of tasks that can be combined with other logic to do complex operations, like run multiple tasks with one command or run certain tasks based on the output of another task. You can also write plans with more refined access control, like allowing users access to run a plan containing a specific task under certain circumstances, but not giving them full access to run the task arbitrarily. Plans can be written in Puppet language or YAML and must be installed using Code Manager.

  The orchestrator uses more memory to support running plans. If your memory usage is near capacity, allocate additional memory before upgrading. See `Java heap` on page 284 for information about configuring Java services.

- **Patch management**
  You can now manage patches on *nix and Windows nodes from the console. After setting up patching node groups, you can view the patch status for your nodes, filter available patches by type and operating system, and run a prefilled task to apply patches to selected nodes.

- **PuppetDB on compilers**
  To improve scalability in large and extra-large installations, Puppet Server and PuppetDB services are now installed by default on new compilers. This change reduces bottlenecks and enables easier scaling, because your PuppetDB capacity is always matched to your server capacity. New compilers installed with the `puppet infrastructure provision compiler` command are configured with the PuppetDB service. After upgrading to this or later versions, you can convert existing compilers with the `puppet infrastructure run convert_legacy_compiler` command.

- **Lockless code deploys**
  Using Code Manager, you can now optionally deploy code to versioned code directories rather than the live code directory. This change enables Puppet Server to continue serving catalog requests even as you deploy code. You can enable lockless code deploys by setting `puppet_enterprise::profile::master::versioned_deploys` to `true`.

  **Note:** Lockless deploys are an experimental feature which might experience breaking changes in future releases. We recommend testing the feature in a non-production environment before enabling it in production.
**Installation and upgrade enhancements**

- PostgreSQL has been upgraded to version 11. For details about upgrading to this version, see [PostgreSQL 11 upgrade in PE 2019.2](#) on page 261.
- Java has been upgraded to version 11. If you customized PE Java services, or use plug-ins that include Java code, test this version thoroughly in a non-production environment before upgrading.
- The `puppet infrastructure upgrade compiler` and `puppet infrastructure upgrade replica` commands let you upgrade compilers or your replica from your primary server. You can specify whether to upgrade one, some, or all of your compilers, and upgrade of multiple compilers occurs at the same time, rather than sequentially.
- The PuppetDB subcommand, `delete-reports`, speeds upgrade by stopping PuppetDB and deleting all reports and their resource events. You can also now use the console to set `resource-events-ttl`, which determines how long PuppetDB stores a report's resource events.
- For Windows agents, the agent install script automatically downloads a tarball of plug-ins from the primary server before the agent runs for the first time. Depending on how many modules you have installed, bulk plug-in sync can speed agent installation significantly.

**puppet infrastructure enhancements**

- Most `puppet infrastructure run` commands no longer require configuring SSH as a prerequisite. The commands that affect PuppetDB, including `migrate_split_to_mono` and `enable_ha_failover`, use the primary server's PuppetDB certificate for authentication. The commands for PostgreSQL cleanup after upgrade; legacy compiler conversion; and compiler, replica, and agent certificate regeneration now use the orchestrator. A new log file located at `/var/log/puppetlabs/installer/orchestrator_info.log` contains run details about these commands.

**Note:** In cases where you're regenerating certificates after recreating your certificate authority, you must still specify SSH connection details, because nodes can't connect to the orchestrator without a valid certificate.

- Certificates for Windows agents can now be regenerated with a single command, `puppet infrastructure run regenerate_agent_certificate`.
- When regenerating agent or primary server certificates, you can now pass an optional `dns_alt_names` parameter.
- The `puppet infrastructure run regenerate_agent_certificate` command includes a `clean_crl` parameter. Setting `clean_crl` to `true` cleans up the local CRL bundle. When you regenerate certificates for *nix agents after recreating your certificate authority, you must include this parameter with the value set to `true`. If you're regenerating agent certificates without recreating the CA, you don't need to clean up the CRL.
- Setting up control repositories for Code Manager no longer requires manually creating an SSH directory and configuring permissions on the key pair and directory. These steps have been automated with the `puppet infrastructure configure` command.
- The `puppet infrastructure status` command now reports status for PostgreSQL as well as the Bolt and Agentless Catalog Executor (ACE) services.

**Disaster recovery enhancements**

- We've changed the way we refer to failover capabilities within PE, from *high availability* to *disaster recovery*. This change more accurately reflects the role of a replica in PE installations, and aligns our terminology with industry standards.
- A replica can now be provisioned up to 3,600 percent faster due to a more efficient method for syncing PuppetDB. Additionally, you can now enable a replica at the same time that you provision by using the `--enable` flag with `puppet infrastructure provision replica`.
- In disaster recovery installations, the `server_list` setting now takes precedence over the `server` setting, preventing unexpected behavior if you promote a replica.
Task, plan, and Puppet run enhancements

• When you run tasks or plans through the orchestrator API or the console, default values are supplied to the job when the user does not supply them. Parameters that have a default value are treated as optional in the console.
• Use the SSH or WinRM protocols to run tasks on target nodes that don’t have an agent installed.
• Use the new task targets API to fine-tune task permissions automatically.
• plan:run and plan:run:* (all plans) allows more control over permissions in plans.
• Schedule a job to run a recurring task in the console or with the /command/scheduled_task endpoint.
• View and delete scheduled jobs with the /scheduled_jobs endpoint.
• View plan details and information about plan jobs you have run using the /plan_jobs endpoint.
• Exclude the plan results from the GET /plan_jobs endpoint with the results parameter.
• Write a single task that has a Python or Bash implementation for Linux and a PowerShell implementation for Windows.
• The following options were added to the orchestrator POST /command/deploy endpoint:
  • filetimeout
  • http_connect_timeout
  • http_keepalive_timeout
  • http_read_timeout
  • ordering
  • skip_tags
  • tags
  • use_cached_catalog
  • usecacheonfailure
• The orchestrator now contains a new JRuby pool to facilitate plans in PE. New configuration options are now available as orchestrator settings to provide control over the JRuby pool.
• There is a new endpoint in the orchestrator that retrieves an individual event, with no truncation of the out_message. See GET /plan_jobs/:job-id/event/:event-id for more information.

Console and console services enhancements

• Navigation and page changes:
  • The Inspect section is now Enforcement.
  • The Run section is now Orchestration.
  • The Overview page is now Status.
  • Puppet runs no longer have a dedicated landing page in the sidebar. Instead, select Run puppet on the Status, Reports, and Jobs pages. The Jobs page is now reserved exclusively for Puppet run results.
  • The Task and Plan pages route to their own details page for status information about scheduled, running, or previously run tasks and plans. This change allows you to run a task or plan and view status information about a task or plan run without leaving that section of the console.
  • The Inventory page has a new look. Instead of using widgets to install agents, nodes, and devices, each installation type has a button that links to its own page. There are also more help icons and definitions on the page.
  • The Run button is now on the Overview, Events, and Packages pages and you can use it to run either a task or Puppet on nodes.
  • Pages in the console now have breadcrumbs, showing you where you are in the interface. The breadcrumbs are links you can use to move to parent pages.
• New task, plan, and Puppet run options and features
  • Run Puppet or a task on a single node from the node details page.
  • Add your own custom PQL queries to the console to run tasks and Puppet on nodes that meet specific conditions. See Add custom PQL queries to the console on page 302.
  • In addition to no-op, you can now specify debug, trace, and eval-trace run options when running Puppet.
  • You can now specify scheduled tasks and Puppet jobs to run every two weeks or every four weeks.
  • When you run a task or a plan in the console, you can now specify an environment other than production.
  • Test connections for nodes and devices before adding them to your inventory. This option is enabled by default on the Inventory page. If a connection fails, you can edit the node or device information and try again.
  • Track node usage and see details about the total number of active nodes in your deployment on the License page.

Classifier enhancements
• Using an updated classification endpoint for the v2 node classifier API, you can now generate an array of the groups that a node was classified into and sort the list by group name and ID. A new top-scope variable is also available, $pe_node_groups.

Code management enhancements
• Setting up control repositories for Code Manager no longer requires manually creating an SSH directory and configuring permissions on the key pair and directory. These steps have been automated with the puppet infrastructure configure command.
• Two new keys in the Code Manager /deploys endpoint provide greater flexibility in deploying modules.
  • "deploy-modules" indicates whether Code Manager deploys modules from the selected environments.
  • "modules" enables you to list specific modules to deploy within the selected environments.

Platform enhancements
• Puppet Server now uses an intermediate certificate authority architecture by default. When you update to a PE version with this capability, you have several options for whether and when to adopt the new intermediate certificate architecture. For details, see Certificate architecture and handling in PE 2019.0 on page 261.
• The puppetserver ca command replaces the puppet cert command, providing a simplified command line interface that uses the new default intermediate CA architecture. Additionally, the new puppet ssl command helps manage certificates on Puppet agents.
• Puppet Server can now run in multithreaded mode, which allows multiple requests to be processed in parallel by a single JRuby. This mode reduces the memory and CPU requirements of the server when enabled.
• Upgrades to PE 2019.1, 2019.3, and 2019.7 involve database migrations that can slow upgrade significantly. Deleting PuppetDB reports and truncating the resource events table before you upgrade can reduce migration time and lessen your downtime. For details, see PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7 on page 259.
• The puppet node purge command now removes all data for the node from PuppetDB. Previously, the command deactivated the node but didn't delete it, so the node could still appear in PuppetDB queries.

Terminology changes
• We've unified infrastructure terminology across all installation types. We now call compile masters compilers to reflect their role: compiling catalogs. Similarly, we call the master a primary server, whether or not your installation includes compilers. In high availability installations, the node that replicates data from the primary server is simply a replica. R.I.P. master, MoM, master of masters, and primary master replica.
• We've updated the terms we use to talk about PE reference architecture types, and are no longer using the term "monolithic."

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Nodes</th>
<th>Former name</th>
<th>New name</th>
</tr>
</thead>
<tbody>
<tr>
<td>All infrastructure components on the primary server.</td>
<td>Up to 4,000</td>
<td>Monolithic</td>
<td>Standard</td>
</tr>
<tr>
<td>All infrastructure components on the primary server, plus one or more compilers and a load balancer.</td>
<td>4,000 to 20,000</td>
<td>Monolithic with compilers</td>
<td>Large</td>
</tr>
<tr>
<td>All infrastructure components on the primary server, plus one or more compilers and a load balancer, plus a separate node which hosts the PE-PostgreSQL instance.</td>
<td>More than 20,000</td>
<td>Monolithic with compilers and standalone PE-PostgreSQL</td>
<td>Extra large</td>
</tr>
</tbody>
</table>

• We've changed the way we refer to failover capabilities within PE, from high availability to disaster recovery. This change more accurately reflects the role of a replica in PE installations, and aligns our terminology with industry standards.

**Changes to default settings**

• TLSv1 and TLSv1.1 is now disabled by default to comply with security regulations. You must enable TLSv1 to install agents on AIX, CentOS 5, RHEL 5, SLES 11, Solaris 10, and Windows Server 2008 R2.

• The default setting for strict_hostname_checking in PE was changed to true to resolve a security issue. You must specify the fully qualified domain name of the host when referring to nodes; partial hostname matches, for example node 'foo' are no longer supported.

• Authentication tokens now have a default lifetime of one hour, instead of the previous lifetime of five minutes. This increase accommodates longer-running puppet infrastructure run commands that use authentication tokens.

• The Puppet CA is now configured to allow signing certificates with authorization extensions by default. This change enables easier setup of new compilers. Signing certificates that have authorization extensions is still disabled in the console, however the puppetserver ca CLI displays authorization extensions in its list output, providing all relevant details about certificates you're signing.

• The shared_buffers setting uses less RAM by default due to improvements in calculating PostgreSQL settings.

**Deprecations and removals**

• MCollective and ActiveMQ have been removed. If you're upgrading from a 2018.1 installation with MCollective enabled, you must take additional steps to ensure a successful upgrade. For details, see MCollective removal in PE 2019.0 on page 261.

• Split and large environment installations, where the primary server, console, and PuppetDB were installed on separate nodes, are no longer supported, and you can't upgrade to this version with a split install. For instructions on migrating to a standard installation, see Migrate from a split to a standard installation in the documentation for your current version.

• Legacy compilers without the PuppetDB service are deprecated. You can convert legacy compilers after upgrading.

• Razor has been removed.

• The node graph in the console has been removed. To generate a node graph, use the Puppet VS Code extension.
• bolt.yaml settings are no longer supported in plans. The modulepath setting is only configurable from environment.conf.

• Many types were moved out of the Puppet codebase, and into modules on the Forge. This change enables easier compositability and reusability of the Puppet codebase and enables development to proceed more quickly without risk of destabilizing the rest of Puppet. See the Resource Type Reference page for the full list and links to the Forge modules for those types that moved.

• Java arguments on all Trapperkeeper services are now fully managed by the puppetenterprise module, and use of pe_ini_subsettings to manage Java arguments is no longer supported. If you add Java arguments to the java_args environment variable using pe_ini_subsettings, the arguments are removed during the next Puppet run.

• The pe_accounts module has been removed. The module was used internally to manage PE users but was superseded several versions ago by the puppetlabs/accounts module. If you’re using the pe_accounts module for account management, migrate to the puppetlabs/accounts module.

• These deprecated parameters have been removed:
  • mcollective_middleware_hosts
  • mcollective
  • mcollective_middleware_port
  • mcollective_middleware_user
  • mcollective_middleware_password
  • activity_database_user
  • classifier_database_user
  • orchestrator_database_user
  • rbac_database_user
  • dashboard_port
  • dashboard_database_name
  • dashboard_database_user
  • dashboard_database_password

Platform release notes
PE uses certain components of open source Puppet. Applicable platform release notes are collected here for convenience.

<table>
<thead>
<tr>
<th>Open source component</th>
<th>Version used in PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet and the Puppet agent</td>
<td>6.28.0</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>6.20.0</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>6.22.1</td>
</tr>
<tr>
<td>Facter</td>
<td>3.14.24</td>
</tr>
</tbody>
</table>

• Puppet release notes on page 74
These are the new features, resolved issues, and deprecations in this version of Puppet.

• Puppet known issues on page 137
These are the known issues in this version of Puppet.

• Puppet Server release notes on page 138
These are the new features, resolved issues, and deprecations in this version of Puppet Server.

• Puppet Server known issues on page 148
These are the known issues in this version of Puppet Server.

• PuppetDB release notes (link)
These are the new features, resolved issues, and deprecations in this version of PuppetDB.
Puppet release notes

These are the new features, resolved issues, and deprecations in this version of Puppet.


Important: Before upgrading, read the upgrade cautions found at https://puppet.com/docs/puppetdb/7/release_notes.html#upgrade-cautions.

Puppet 6.28.0
Released August 2022.

Enhancements

Extend Package Source to include HTTP URLs (Windows)

Puppet now supports installing .exe packages using https as a package source. For example:

```puppet
package { 'mysql':
  ensure => '5.5.16',
  source => 'https://example.com/packages/mysql-5.5.16-winx64.exe',
  install_options => ['INSTALLDIR=C:\mysql-5.5'],
}
```

PUP-11502

Bump to openssl-fips-1.1.1k-6

Updated openssl-fips on Red Hat to 1.1.1k-6. PA-4498

Update puppet-ca-bundle

Updated root certificate authority bundle included with puppet-agent. PA-4496

Support for macOS 12 (M1)

This release adds support for macOS 12 (M1). PA-4457

Support for Windows 11 Enterprise (x86_64)

This release adds support for Windows 11 Enterprise (x86_64). PA-4249

Support for Ubuntu 22.04 (x86_64)

This release adds support for Ubuntu 22.04 (x86_64). PA-4233

Resolved issues

Puppet::HTTP::Client cannot connect to a server requiring client cert authentication and whose server cert is issued by a CA in the ssl_trust_store

Puppet's http client can now establish a mutually authenticated TLS connection when passing include_system_store: true such as when retrieving file content from HTTPS servers. Previously puppet
did not add its client certificate to the SSL context, so the connection would fail if the HTTPS server required a client certificate. PUP-11522

**Legacy function error does not include the source ref**

If a 3x function produces an error, the error message now includes the path to the file in which the function is defined. Contributed by Puppet community member lolipopman. PUP-11472

**puppetserver_gem doesn't install gems when they are loaded by Facter**

Fixed a bug that prevented the puppetserver_gem provider from managing gems that were first loaded by Facter. PUP-11452

**Puppet Agent does not automatically refresh CRLs on crl_refresh_interval**

Puppet Agent now reloads its CA and CRL bundles every 30 minutes during each run. Previously it only loaded it when the process started, which meant the service had to be restarted if the CA/CRL files changed on disk. PUP-11428

**Puppet agent --disable is ignored with cron puppet agent (splay).**

Puppet agent now checks the disabled lock file after sleeping due to splay. PUP-9998

**puppet-cacerts keystore is missing on Red Hat 9, SLES 15 and Ubuntu 20.04**

If Puppet agent is installed, there is a java keystore file. PA-4440

**Deprecations and removals**

**Removed EOL operating system support**

This release removes support for the following operating systems: Fedora 32, CentOS 8, Ubuntu 16.04. PA-4328

**Security**

**Update puppet runtime's curl to 7.83.1**


**Puppet 6.27.1**

Released April 2022.

**Enhancements**

**Allow Puppet::HTTP::Client to connect to trusted server using the puppet certificate for client authentication**

You can now specify an https URL as the source of a file resource when the TLS server requires a client certificate for authentication. PUP-11471

**Resolved issues**

**Ruby security fix**

Applied patch for security vulnerability in Ruby 2.5.9 (CVE-2022-28739). PA-4364

**Puppet::Util::Windows is undefined on non-Windows platforms**

Fixed a bug that prevented pdk unit tests from working when trying to test a resource with a Windows provider, such as "service" resources. PUP-11459
No option to fail fast when agent-specified environment does not exist
When using strict_environment_mode=true, a run now fails early if the requested environment does not exist on the server, or if the server does not allow the agent to specify its own environment. PUP-11440

puppet lookup fails to interpolate topscope variables when an environment is specified
Fixed an issue where Puppet 6.26 and 7.14 failed to resolve toplevel facts in hiera configs when using the --environment option for puppet lookup. PUP-11437

Rspec tests with custom facts fail on some modules
This release fixes an issue where rspec module tests would compile with the runner node’s facts instead of using the custom facts supplied by the test. PUP-11435

Nokogiri upgrade for macOS
Upgraded nokogiri gem to 1.13.2 on macOS due to upstream security fix. PA-4323

Some gemspecs are missing from puppet-agent MSI
On Windows, it is now possible to install a gem that has a dependency on Facter or Hiera into Puppet's vendored ruby. PA-4313

Puppet 6.27.0
This version was never released.

Puppet 6.26.0
Released January 2022.

Enhancements

ENC-enforced environment bypass for lookup
You can now bypass the ENC-enforced environment when performing a lookup. To bypass the enforced environment, use lookup with the --environment option to specify the desired environment. Puppet always uses the environment you specified regardless of the ENC-enforced environment. PUP-7479

Support for RHEL 9
This release includes support for Red Hat Enterprise Linux (RHEL) 9. PUP-11364

Support for Windows Server 2022(x86_64)
This release includes support for Windows Server 2022(x86_64). PUP-11238

Resolved issues

Failures when using purge_ssh_keys
Puppet no longer fails when using purge_ssh_keys if the user's home directory doesn't exist yet. PUP-11380

Puppet::FileSystem.chmod does not validate its arguments
Puppet::FileSystem.chmod now validates its arguments like other methods. PUP-11345

Warning: #<Puppet::Transaction::Persistence after upgrading to Puppet agent 6.25.0
Fixes a regression introduced in 6.25.0 and 7.10.0 that caused a Puppet::Transaction::Persistence warning during each agent run. PUP-11321
**User resource tries to create rather than modify users created by a utility**

This release moves the `ssh_authorized_key` resource's creation to the end of the user type flow, after all user properties and parameters were resolved, to avoid order dependency errors. PUP-11320

**Puppet code merger using incorrect command**

Reduces memory usage when parsing manifests. PUP-11318

**Failure when using the names "apply" and "plan" within an apply() block in a plan**

The names "apply" and "plan" can now be used as resource parameter names in all cases. Previously, using them within an apply() block in a plan would fail. PUP-11315

**Puppet attempts to execute directories from /etc/init.d/**

Prevents Puppet from considering directories from /etc/init.d/ as services. PUP-11313

**High memory consumption from lib/puppet/pops/parser/lexer2.rb**

Reduced lexer2 memory usage. PUP-11236

`versioncmp()` treats 11.0 as greater than 11

`versioncmp()` now strips redundant numbers. PUP-11235

**puppet lookup --facts {filename} fails if filename does not contain a dot**

Before this release, `puppet lookup --facts {filename}` failed early when the filename given did not contain a dot. This fix removes the early extensions check and adds a fallback instead: tries both formats (JSON then YAML) to read the given facts file when its path doesn't end with any of the expected extensions (yaml/yml/json). Otherwise, it follows previous implementation and respects the given extension. PUP-11204

**Facts provided in a file cannot be used for classification**

Fixed a bug where facts provided in a file were not being merged with the facts used for classification. This is because Puppet collected and merged the said facts after the classification happened. To fix this, we ensured that Puppet resolves the facts being used for classification before the node request. PUP-10435

**Inconsistent handling of trusted facts in the lookup CLI**

When using `puppet lookup with --facts`, if the facts file overrides any of hostname, domain, fqdn, clientcert, then it must override all of them. Also, if a value for certname is provided in a fact file for the lookup application, use it when creating the trusted information object. This makes it possible to override trusted.certname for classification. PUP-8220

**Lookup ignores environment from the classifier when using a rule with trusted facts**

Fixed an issue where trusted facts could not be used as rules for classification. This was fixed by gathering the trusted facts from the PuppetDB query result, and overriding the trusted facts context. PUP-8094

**Misleading results when using --node flag in puppet lookup**

Fixed an issue where `puppet lookup` would result in misleading results when using the --node flag. This happened because there can be cases where the target node does not have any facts cached. To avoid this, the fix implemented checks for the node facts/facts given in a fact file, and if it doesn't find any it raises an error. PUP-7362
Puppet could not retrieve attributes from fifo and socket files
This release allows Puppet to retrieve attributes for fifo and socket files and manage them when the given manifest has a file resource which is recursing over a given path. PUP-4045

Puppet hangs trying to replace a FIFO
Puppet no longer hangs when trying to replace a fifo with a file, directory or symlink. PUP-1460

Puppet uses deprecated psych features
Puppet is now compatible with psych 4.0. PUP-11405

Agent no longer calls the Puppet::Node terminus to resolve the environment during the run
Introduced a Puppet setting use_last_environment=true|false and a corresponding puppet agent -t --no-use_last_environment boolean command line option that forces the agent to make a node request like it did prior to 7.12 and 6.25. By default, the agent does not make a node request. PUP-11379

Puppet user and service resources are slow on Mac OS X
Managing users and services on macOS is much faster. PUP-11332

Puppet::Node#environment_name may return the wrong value
Puppet::Node#environment_name now always returns the symbolic name of the environment (if one has been set on the node). PUP-11330

Puppet lookups failed due to missing certificates
The puppet lookup command now works if the agent does not have certificates available locally. PUP-11402

The generate types command does not handle errors correctly
If the generate types command failed to generate a custom type, it logged an error and returned a 0 exit code instead of failing. The command now correctly fails with a non-zero exit code if the command cannot generate a type. PUP-11078

Bump Ruby date gem
We've bumped the date gem in our vendored Ruby to fix a vulnerability. PA-4131

Puppet 6.25.1
Released November 2021.
Enhancements

Faster iterative functions
This release speeds up the amount of time it takes to type check arguments passed to blocks of iterative functions, such as reduce and merge. PUP-9561

Resolved issues

Puppet can leak credentials when following HTTP redirects
Previously, Puppet followed HTTP redirects, the Authentication and Cookie headers were passed to different hosts, which could leak sensitive information. Now the Authentication and Cookie headers are only sent when redirecting to the same hosts. This fixes CVE-2021-27023. PUP-11188
Puppet agent silently skips unknown resources
Previously, all unknown resources were converted into a component (Puppet::Type::Component) by default and skipped when applying a catalog. This release adds a new resource attribute that specifies the type of resource — this is used to differentiate between built-in types and user defined types. Resources that are known and available on the server node are also verified on the agent node, which now fails when something unknown is found in the received catalog. This fixes CVE-2021-27025. PUP-11209

Note: Puppet Agent 6.25.1 and 7.12.1 introduced a new catalog resource field in order to resolve CVE-2021-27025. This field was not handled properly by older versions of PuppetDB and results in catalogs not being stored in PuppetDB. Before upgrading any agents in your installation to 6.25.1 or 7.12.1, you must first upgrade your PuppetDB(s) to this version. (PDB-5338)

Puppet gem and rspec-puppet failures
This release moves the DEFAULT_TIMEOUT constant from lib/puppet/util/windows/service.rb to lib/puppet/util/windows.rb in a non-OS guarded code area. This change avoids uninitialized constant errors when compiling catalogs on non-Windows operating systems. PUP-11319

Puppet agent downloads all plugins after updating
Puppet 6.25.0 and 7.12.0 introduced a regression which caused a newly upgraded agent to download all of its plugins. Now the agent performs a single node request to resynchronize its environment with the server. PUP-11328

Puppet 6.25.0
Released October 2021.

We would like to thank the following Puppet community member for their contributions to this release: natemccurdy.

New features
The write-catalog-summary setting
This release adds the write_catalog_summary setting to control whether the resources.txt and classes.txt files are written to disk after applying a catalog. By default, puppet agent and puppet apply behave the same as before — puppet agent writes the files, and puppet apply does not. PUP-1042

Enhancements
Support for Ubuntu 18.04 (aarch64)
This release adds support for Ubuntu 18.04 (aarch64). PUP-11162

Support for AlmaLinux 8 (x86_64)
This release adds support for AlmaLinux 8 (x86_64). PUP-11242

Support for Rocky Linux 8 (x86_64)
This release adds support for Rocky Linux 8 (x86_64). PUP-11231

Puppet module type scripts directory
This release adds a new scripts/ subdirectory to the module class. It automatically generates the scripts? (), scripts(), and script() functions in the class for retrieving the available scripts. This helps to standardize specific file loading from either the files directory or scripts directory in a module. PUP-11165
Lower memory consumption in Ruby files
This release lowers memory consumption by 10%. This is achieved by freezing string literals in Ruby classes with high memory consumption. PUP-11232

Facter.value replaced by Puppet.runtime[:facter]
This release replaces calls to Facter.value with calls to Puppet.runtime[:facter], and removes require 'facter' statements. PUP-11217

Support for multiple Facter implementations
You can now register a Facter implementation when initializing Puppet via the Puppet.initialize_settings method. PUP-11216

Load Task files from scripts
Tasks can now load files from the scripts mount. PUP-11200

onlyif and unless in --noop documented
This release documents the noop behavior of the onlyif and unless parameters of the exec resource. PUP-11199

Scripts file serving mount
When using Puppet APIs to load file content and metadata, you can access files in the scripts/ directory of a module using the scripts file mount. PUP-11187

RHEL9 support for services
Puppet now uses systemd as the default service provider for EL 9 variants, such as the Red Hat or CentOS stream. PUP-11168

The launchd service provider fails if a parsable but invalid LaunchAgent or LaunchDaemon plist file exists
This release fixes an issue where the launchd service provider failed if a parsable but invalid LaunchAgent or LaunchDaemon plist file existed. PUP-11164

Cleaned up ext/ directory
This release removes unused files from the ext/ directory used by upstream Linux and Solaris packages. PUP-10685

Option to enable long filename support in the Windows MSI installer
This release updates the MSI installer for Puppet agents to enable long filenames either through a check box in the installer or by setting the ENABLE_LONG_PATHS=true option in the command line. PA-3843

OpenSSL updates
- On Windows 2012 FIPS (x64), OpenSSL 1.0.2 has been patched for CVE-2020-1971, CVE-2021-23839, CVE-2021-23840, CVE-2021-23841 and CVE-2021-3712 - PA-3976
- On redhatfips-7-x86_64 RedHat 7 FIPS (x64), OpenSSL has been bumped to 1.1.1k and patched for CVE-2021-3712 and CVE-2021-3711 - PA-3974

On all other platforms, OpenSSL has been bumped to 1.1.11 - PA-3925
Deprecations and removals

Support for Fedora 30 (x86_64) removed
This release removes support for Fedora 30 (x86_64). PUP-11092

Support for Fedora 31 (x86_64) removed
This release removes support for Fedora 31 (x86_64). PUP-11093

Support for Mac OS X 10.14 (x86_64) removed
This release removes support for Mac OS X 10.14 (x86_64). PUP-11094

Support for Red Hat 5 (x86_64, i386) removed
This release removes support for Red Hat 5 (x86_64, i386). PUP-11091

Resolved issues

Default timeout ignores Windows services
Previously, default timeouts caused issues on Windows when services took longer than 10 seconds to change state. You can now specify the default timeout value for syncing service properties. PUP-10925

User attributes ignores forcelocal
This release fixes an issue where setting forcelocal => true on an user resource checked the resource's home and shell attributes against their values from the directory service provider. Contributed by Puppet community member natemccurdy. PUP-11241

Puppet fails to install packages on Solaris if another pkg install is running
Installing packages on Solaris with the pkg command does not work if another instance of pkg is already running. Now Puppet tries the install command 5 times, and only fails if the package cannot be installed. PUP-11208

The concat module ignores the ENC environment
This release fixes an issue where an ENC-specified environment was not pushed during a Puppet run. This caused indirector requests with no specified environment to default to using an incorrect environment. PUP-11265

Util::JSON.dump receives non-hash options
Previously, Puppet's /puppet/v3/file_metadatas REST API failed if the multi_json gem was uninstalled or when it was running puppetserver from source. PUP-11237

Puppet sends warning for BOM and US-ASCII encoding
This release removes BOM for non-UTF encoding and its warnings. ASCII characters are single bytes, which means there is no need for a BOM to detect byte ordering (LSB/MSB). PUP-11196

User resource not removing password on AIX agents
This release fixes an issue where deleting an AIX user with Puppet would not clean up the user's password. PUP-11190
Undefined method [] for nil:NilClass when handling SemanticPuppet::Dependency::UnsatisfiableGraph

Previously, the puppet module install command broke when dependencies could not be resolved. Puppet now emits an error message instead. PUP-11172

User resource unable to remove the home directory
This release fixes an issue where the user home directory was not removed when managehome was set to true. PUP-11170

Static catalogs not working for file resources when versioned_deploys is enabled

Previously, when :versioned_environment_dirs was set to true, catalog compilation failed to add metadata for static catalog file resources; this meant that an agent receiving a catalog would not attempt to request that static file content. This has been fixed and the metadata is now correctly added to the catalog when :versioned_environment_dirs is set to true. PUP-11169

Changes to current working directory when listing modules

Puppet Server and agent no longer change their current working directories when listing modules in an environment directory. PUP-11166

An environment reloaded during a single compilation could fail

Previously, Puppet Server could reload an environment while it was being used to compile a catalog. If translations were enabled (Puppet[:disable_i18n] set to false), compilation could fail. Now Puppet Server prevents environments from being reloaded while they are in use, and instead reloads the environment the next time it is requested. PUP-11158

The pkg provider cannot unhold and update packages in the same run

Previously, the pkg package provider was unable to handle manifests where a package was updated and marked as unhold at the same time. This is now fixed. PUP-10956

Environment caches string and symbol environment names differently

This release fixes an issue that resulted in Puppet caching duplicate copies of an environment. PUP-10955

A lookup fails if lookup_options is empty

Previously, when lookup_options were defined at the global or environment layer, and the module defined an empty hash, the compilation failed. This is now fixed and the empty hash is ignored. PUP-10890

Rich data types can corrupt the transaction store

This release fixes an issue that prevented Puppet from reporting corrective changes when using rich data types such as Deferred, Binary, and Sensitive. PUP-10820

Puppet.lookup(:current_environment) is wrong if the environment changes during convergence

This release fixes an issue where an old environment could be used if the environment had changed due to pluginsync. PUP-10308

Failure to fetch node definition results in bad pluginsync and cascading failure

Previously, Puppet agents would make a node definition request to the server to find out the correct environment to run in. This request has now been removed, and the agent saves its last used environment in the last_run_summary.yaml file. If the environment is not set in the CLI or config, agents attempt to use the
Environment in last_run_summary.yaml — only if the previous run had an agent/server environment mismatch. PUP-10216

**The puppet resource --to_yaml emits class tags**

This release stops the resource --to_yaml command emitting Puppet class tags, such as Puppet::Util::Execution::ProcessOutput, by ensuring that the PScalarDataType only checks the instance of String, and not other subclasses. PUP-10105

**Catalog failure on first run due to pluginsync and environment switch**

Previously, an agent failed its run if it switched to a new environment where the manifests relied on a fact that only existed in the new environment. Now the agent redirects to the server-specified environment and the run continues using that environment. PUP-9570

**Agent cannot compile catalog if it specifies an non-existent environment in puppet.conf**

This release fixes an issue that caused the agent run to fail if the agent requested an environment that did not exist on the server — even when the classifier controlled the environment. PUP-6802

**Puppet agent does not save local copy of last_run_report.yaml**

The agent now saves a local copy of its last run report, even if it fails to submit the report to the primary Puppet Server. PUP-6708

**User resource exposes hashed password when changing password or adding a user**

Previously, when managing passwords with the useradd provider, the password hash appeared when listing running processes. Now the password is set with the chpasswd command that uses stdin to receive the password from a temporary file, so it no longer appears in the process list. PUP-3634

**Puppet 6.24.0**

Released July 2021.

We would like to thank the following Puppet community member for their contributions to this release: cocker-cc

**Enhancements**

**The puppet ssl show command prints custom object identifiers (OID)**

The puppet ssl show command now shows the names of certificate extensions containing custom OIDs — when the trusted_oid_mapping_file exists. This functionality used to exist in the puppet cert print command. PUP-11120

**Updated argument error message**

If you call a function with an argument Puppet does not accept, the error message provides a list of acceptable function signatures. PUP-7792

**Updated error message for incorrect module name**

If the author component of a module name is omitted, the puppet module install <author-module> command provides a name suggestion in the error message. PUP-10641

**Puppet reports the license gem on Apache**

Puppet now reports the Apache 2.0 license when installed as a gem. PUP-11118

**Support for Debian 11 Bullseye amd64**

This release adds support for Debian 11 Bullseye amd64. PUP-11030

© 2022 Puppet, Inc., a Perforce company
macOS puppet-agent code-signs executables
The macOS puppet-agent AIO packages now provide code-signed executables for puppet and pxp-agent. PA-3756

Solaris OpenSSL patching replaced with compiler arguments
This release adds AES CTR-DRGB performance improvements to Puppet’s vendored OpenSSL. PA-3698

The empty function accepts Sensitive data types
The empty function now accepts Sensitive data types, which allows you to test a Sensitive variable that is neither nil or empty. For example, a variable in an ERB template. Contributed by Puppet community member cocker-cc. PUP-11124

The unwrap function accepts Any data type
The unwrap function now accepts the Any data type. This means that the component modules, such as puppetlabs-postgresql, can migrate to using Sensitive values, while still accepting non-Sensitive values. You do not need to special case when unwrapping the value. Contributed by Puppet community member cocker-cc. PUP-11123

The exec provider supports commands as an Array
When a command is an Array of Strings, passed as [cmdname, arg1, ...], it is now executed directly instead of being passed to the standard shell. This is supported for the following exec parameters: comand, onlyif, unless, refresh. Note that onlyif and unless already accept multiple commands as an Array — you need to pass the value as an Array of Array to use this new behaviour. PUP-5704

Embedded Ruby (ERB) templates allow a leading Byte Order Mark (BOM)
Previously, when a template contained a BOM, it was preserved by the template function and included in the resulting file or PowerShell command. Puppet now passes the bom option when reading the file, removing the BOM as it is read. PUP-8243

Deprecations and removals

Support for Debian 8 removed
This release removes support for Debian 8. PUP-11059

Resolved issues

The --extra cli option is not functional
The puppet help command no longer displays the --extra command line option. PUP-8700

The parsedfile provider produces an undefined method each for nil::NilClass
Puppet now prints an error if a parsedfile provider returns nil, for example, when using the nagios_core module. PUP-9369

Unclear error message if user or group providers are not suitable
Puppet now prints a more detailed error message if the user or group providers are not functional. PUP-9825

The Puppet::Resources.search method fails when conditions are provided
This release fixes an issue that prevented the Puppet::Resource.indirection.search method from accepting conditions when filtering results. PUP-7799

© 2022 Puppet, Inc., a Perforce company
Repository error message URL is missing part of the path
Previously, the puppet module command reported an incorrect URL in the error message when the module_repository setting was overridden. This is now fixed. PUP-8650

The desired_value file mode is reported without leading zeros
Puppet now reports file modes with a leading zero in the desired_value field — for example, 0755 — which is consistent with the previous_value. PUP-7493

Filebucket fails when using a non-default environment from the server
The filebucket application no longer requires an environment to exist locally. PUP-10796

Unable to load PKey.read with private keys
Puppet agent now loads private keys in the PKCS#8 format. PUP-11082

Cached environments are not deleted when the directory is removed
Puppet now removes environments that are no longer on disk. PUP-11129

Unable to run the puppet resource command when the environment is specified
Previously, running puppet resource on the agent with an invalid environment would fail. With this release, the application falls back to the default environment, if the specified one does not exist. PUP-6554

Puppet prints unnecessary errors in debug
Puppet no longer prints an unnecessary error message when resolving account names to security identifiers on Windows. PUP-10967

Setting age=0 on a tidy resource does not remove all files
Previously, the age parameter of the tidy resource only removed files older than those specified. This is now fixed and Puppet removes all files. PUP-11079

The agent_specified_environment fact is not populating
This release fixes an issue where the agent_specified_environment fact did not populate when the environment was set in the [agent] section. This is now fixed and populates in the following order: CLI, agent section, main section. PUP-6801

pip ensure=>latest fails with pip>=20.3.0
In version 20.3b1, pip removed the ability to list available versions of a package. This release adds the --use-deprecated=legacy-resolver argument so that you can query available versions. PUP-11029

The pxp-agent does not use the wrapper script
The pxp-agent service script on the AIX, OSX, and Solaris platforms now manipulates the service using the wrapper script located in /opt/puppetlabs/bin/pxp-agent, which cleans up the linker environment before calling the actual environment. This prevent failures due to incompatible libraries being loaded. To modify the pxp-agent linker environment, directly call the pxp-agent binary, for example, /opt/puppetlabs/puppet/bin/pxp-agent. PCP-890

Puppet 6.23.0
Released June 2021.

We would like to thank the following Puppet community member for their contributions to this release: tobias-urdin.
Enhancements

Fedora 34 support
This release adds support for Fedora 34. PUP-10892

File limit with the max_files parameter
By default, the file and tidy resource types generate a warning on the Puppet Enterprise (PE) console and report when Puppet tries to manage more than 1000 files with the recurse parameter set to true. The file and tidy resource types now support a new parameter — max_files — that enforces a hard limit. If the number of recursive files is greater than the limit, the agent run fails. You can set the max_files parameter to -1 to disable the warning. PUP-10946

Case sensitive parameter for the fqdn_rand() function
The fqdn_rand() function now accepts an optional parameter to downcase the FQDN fact, so that the function's result is not case sensitive. You must pass the parameter after the seed string, for example, fqdn_rand(100, 'expensive job 1', true). By default, the function remains case-sensitive. PUP-10922

Improve enable=delayed_start error message
This release improves an error message to properly convey that you cannot set a systemctl service to delayed_start on operating systems other than Windows. PUP-11062

Ruby support long paths on Windows
This release adds the following patch into the Puppet Agent vendored Ruby. The patch implements long path support on Windows. PA-3759

Improved Ruby Performance
New Ruby performance patches — reducing 50-90% of file IO when loading Puppet and Facter. PA-3526

Bump curl to 7.77.0
This release bumps the curl dependency to 7.77.0. PA-3762

Resolved issues

PUPPET_SERVER MSI install property does not work
Previously, using PUPPET_SERVER as an MSI property did not set the server setting. This is now fixed. PA-3667

Puppet does not specify SELinux filetype when getting the default context
Previously, Puppet created files with the wrong default SELinux context, which was only corrected after a subsequent Puppet run. This is now fixed. Contributed by Puppet community member tobias-urdin. PUP-7559

NIM provider used very restrictive regular expressions
Previously, the NIM provider only allowed numbers when parsing RPM release tags and didn’t accept bff (installp) packages marked as security updates in the header. In this release, Puppet allows installation of such packages. PUP-3631

Unable to mask a static systemctl service
This release fixes an issue where the systemctl provider did not mask static systemctl services. Contributed by Puppet community member nmaludy. PUP-11034

© 2022 Puppet, Inc., a Perforce company
**Masking service failed**
Previously, Puppet failed to mask a systemd service that did not exist. This bug is now fixed. PUP-10974

**Sensitive instances shared the same value yet weren’t equal**
Previously, two type Sensitive instances failed to compare as equal—despite sharing the same underlying strings. In this release, comparisons such as $a = Sensitive("secret"); $b = Sensitive("secret"); notice($a == $b) now return as true. PUP-11061

**SemVer datatype components failed to pass as hash or argument list**
Previously, the build or prerelease components of the SemVer datatype failed to pass as a hash or list of arguments. This bug is now fixed. PUP-11077

**User keychains were inaccessible to Puppet Agent**
Previously, user keychains were inaccessible to Puppet Agent if you ran Puppet Agent through the macOS daemon. This bug is now fixed. PUP-11081

**Puppet returned an error when specifying the purge_ssh_keys parameter**
Previously, Puppet returned an error if you specified the purge_ssh_keys parameter for a user resource that didn’t previously exist. To fix this bug, Puppet prioritizes the ensure property of a user before the purge_ssh_keys parameter. PUP-11067

**Nil vertices caused resource management errors**
Previously, managing resources that call the generate method—failed when using the puppet resource subcommand—due to the presence of a nil vertex in the catalog. To fix this bug, Puppet can no longer add nil vertices to the catalog. PUP-11074

**Unable to update UserRightAssignment**
Previously, validating the logonaccount and logonpassword parameters for the service resource on Windows failed too early. This release moves the parameters further down the catalog compilation order list to avoid early errors. PUP-10999

**Puppet cannot change/set new user passwords on macOS Big Sur**
Previously, you could not set or change the password of a new user created on macOS Big Sur. This bug is now fixed by ensuring the ApplicationAuthority field exists whenever you create a new user. PUP-11026

**Puppet returned an error when creating new users on macOS 10.14**
Previously, if you created a new user on macOS 10.14, Puppet returned an operation not permitted @ rb_sysopen error. This bug is now fixed. PUP-11095

**Agent failures with server_list**
Previously, when Puppet processed server_list and tried to find a functional server, it threw an error if it could not connect, causing the agent to fail. This is now fixed. PUP-10844

**Puppet 6.22.1**
Released 26 April 2021.

We would like to thank the following Puppet community members for their contributions to this release:
gcampbell12, cschug, and StackKorora.

© 2022 Puppet, Inc., a Perforce company
New features

**Improvements to the puppet facts diff command**

This release makes the following improvements to the `puppet facts diff` CLI command:

- The command shows all of the differences between Facter 3 and Facter 4 — including facts only available in Facter 4 — and sorts the output results in alphabetical order. You can also remove certain facts from the output using the `--exclude <regex>` CLI option and see the results in a fully structured form. [PUP-10985](#)
- The command filters the output based on an exclude list. This only contains volatile facts — those that are expected to change value between consecutive Facter runs. [PUP-10947](#)
- The command pretty-prints JSON output. You can specify the older form with `--render_as json`. [PUP-10973](#)

**Backport logic to detect migrated CA directory location**

After migrating the CA directory, Puppet now reports the correct `cadir` setting value. [PUP-11004](#)

**Curl bumped to 7.76.0**

This release bumps Curl to 7.76.0, fixing the following CVEs:

- [CVE-2021-22890](#)
- [CVE-2021-22876](#)
  [PA-3690](#)

**OpenSSL bumped to 1.1.1k**

This release bumps OpenSSL to 1.1.1k, fixing the following CVEs:

- [CVE-2021-3450](#)
- [CVE-2021-3449](#)
  [PA-3669](#)

**Ruby bumped to 2.5.9**

This release bumps Ruby to 2.5.9, fixing the following CVEs:

- [CVE-2020-25613](#)
- [CVE-2021-28965](#)
  [PA-3695](#)

**The GlobalSignRoot CA R3**

This release adds the GlobalSignRoot CA R3 certificate for rubygems.org. [PA-3525](#)

**The ciphers setting**

The `ciphers` setting configures which TLS ciphersuites the agent supports. The default set of ciphersuites is the same, but you can now make the list of ciphersuites more restricted, for example, to only accept TLS v1.2 or greater ciphersuites. [PUP-10889](#)

**The puppet ssl show command**

The `puppet ssl show` command prints the full-text version of a host's certificate, including extensions. [PUP-10888](#)
The puppet facts show command backported to 6.x agent

The puppet facts show command can now retrieve a list of facts from the system. Run puppet facts show --help to see all of the supported options. PUP-10884

Resolved issues

Puppet cannot parse systemd instances when the list-unit-files output has an additional column

This release fixes an issue affecting the parsing of systemd service instances caused by a change in the systemctl list-unit-files command output. PUP-10949

Race condition with agent_disabled_lockfile

This release fixes a race condition that caused the agent to become disabled and no longer enforce desired state. Contributed by Puppet community member gcampbell12. PUP-11000

Retrieve SID for users under APPLICATION PACKAGE AUTHORITY

A known issue with LookupAccountNameW caused Puppet to fail when managing Windows users under APPLICATION PACKAGE AUTHORITY with fully qualified names. This is now fixed and we have added an account name step to prevent faulty queries. PUP-10899

Retrieving the current user with the fully-qualified username fails on Windows

Previously, retrieving the current username SID on Windows caused Puppet to fail in certain scenarios, for example, when the user was a secondary domain controller. This release adds a fallback mechanism that uses the fully qualified domain name for lookup. You can retrieve the current user with the fully-qualified username and domain on Windows. PUP-10898

Windows package provider continues to read the DisplayVersion key after it is embedded NULL

Previously, Puppet would not stop reading the registry at the correct WCHAR_NULL because it was encoded to UTF-16LE, causing Puppet to read bad data and fail. This is now fixed. PUP-10943

Listing environments during code deploys prevents environment cache invalidation

Previously, catalog compilations for a newly created environment directory could fail if the environment was listed while the directory was being created. This issue only occurred when using an environment_timeout value greater than 0 and less than unlimited. This is now fixed. PUP-10942

User resource with forcelocal and groups attributes set fails if /etc/group contains empty lines

This release fixes an issue where Puppet failed when applying user resources with forcelocal if there were empty lines in /etc/group. PUP-10997

Augeas fails with Postgresql 13

This release patches the vendored Augaes in puppet-agent so that PostgreSQL 13 configuration file is no longer parsed. PA-3406

The puppet facts diff command logs error when the apt module is installed

Previously, the puppet facts diff command, which compares Facter 3 and Facter 4 facts, did not run Facter 4 in a clean environment, causing undefined behavior for custom facts resolving. With this release, Puppet 4 facts are read when running puppet facts in a new shell and custom facts resolve correctly. PUP-10940
Cannot ensure `dnfmodule` with no default profile

Previously, using the `dnfmodule` provider to install a module with no default profile and without passing the `enable_only` parameter failed in newer versions of DNF. [PUP-11024](#).

**Puppet 6.21.1**

Released 16 February 2021.

Resolved issues

**Puppet users with `forcelocal` are no longer idempotent**

This release fixes a regression where setting the `gid` parameter on a user resource with `forcelocal` was not idempotent. [PUP-10896](#).

**Puppet 6.21.0**

Released 9 February 2021.

Resolved issues

User resource with `forcelocal` uses `getent` for groups

The `useradd` provider now checks the `forcelocal` parameter and gets local information on the groups (from `/etc/groups`) and gid (from `/etc/passwd`) of the user when requested. [PUP-10857](#).

Slow Puppet agent run after upgrade to version 6

This release improves the performance of the `apt` package provider when removing packages by reducing the calls to `apt-mark showmanual`. [PUP-10856](#).

The `apt` provider does not work with local packages

The `apt` package provider now allows you to install packages from local file using source parameter. [PUP-10854](#).

**Puppet 6.20.0**

Released 20 January 2021.

New features

**JSON terminus for node and report**

This release implements JSON termini for node and report indirection. The format of the `last_run_report.yaml` report can be affected by the cache setting key of the report terminus in the `routes.yaml` file. To ensure the file extension matches the content, update the `lastrunreport` configuration to reflect the terminus changes (lastrunreport = $statedir/last_run_report.json). [PUP-10712](#).

**JSON terminus for facts**

This release adds a new JSON terminus for facts, allowing them to be stored and loaded as JSON. Puppet agents continue to default to YAML, but you can use JSON by configuring the agent application in `routes.yaml`. Puppet Server 7 also caches facts as JSON instead of YAML by default. You can re-enable the old YAML terminus in `routes.yaml`. [PUP-10656](#).

**Multiple `logdest` locations in puppet.conf accepted**

You can set multiple `logdest` locations using a comma separated list. For example: `/path/file1,console,/path/file2`. [PUP-10795](#).

**New `serverport` setting type**

The `serverport` setting is an alias for `masterport`. [PUP-10725](#).
The **puppet module install** command lists unsatisfiable dependencies

If the **puppet module install** command fails, Puppet returns a more detailed error, including the unsatisfiable module(s) and its ranges. PUP-9176

**Interpolation of sensitive values in EPP templates**

Previously, if you interpolated a sensitive value in a template, you were required to unwrap the sensitive value and rewrap the result. Now the **epp** and **inline_epp** functions automatically return a Sensitive value if any interpolated variables are sensitive. For example: **inline_epp**("Password is <%= Sensitive('opensesame') %>”). Note that these changes just apply to EPP templates, not ERB templates. PUP-8969

**OpenSSL bumped to 1.1.1i**

This release bumps OpenSSL to 1.1.1i. PA-3513

**Curl bumped to 7.74.0**

This release bumps Curl to 7.74.0. PA-3512

**(Experimental feature) puppet facts diff action**

This release adds a new action called **puppet facts diff** to check if there are differences between Facter 3 and Facter 4 outputs. Facts that change over time like **memory** and **uptime** are ignored. PUP-10815

**Deprecations**

**func3x_check setting removed**

The **func3x_check** setting has been removed. PUP-10724

**Resolved issues**

**The puppet apply command creates warnings**

This release eliminates Ruby 2.7.x warnings when running **puppet apply** with node statements. PUP-10845

**Remove Pathname#cleanpath workaround**

This release removes an unnecessary workaround when cleaning file paths, as Ruby 1.9 is no longer supported. PUP-10840

**Cached catalog contains the result of deferred evaluation instead of the deferred function**

Puppet 6.12.0 introduced a regression that caused the result of a deferred function to be stored in the cached catalog. As a result, an agent running with a cached catalog would not re-evaluate the deferred function. This is now fixed. PUP-10818

**Calling scope#tags results in undefined method**

Previously, calling the tags method within an ERB template resulted in a confusing error message. The error message now makes it clear that this method is not supported. PUP-10779

**Puppet agent installation fails when msgpack is enabled on puppetserver**

Previously, the agent failed to deserialize the catalog and fail the run if the **msgpack** gem was enabled but not installed. Now the agent only supports that format when the **msgpack** gem is installed in the agents vendored Ruby. PUP-10772
**Setting facterng twice raises an exception**

Previously, calling `puppet config set facterng true` twice resulted in a failure. This is now fixed and `facterng` is only enabled if it is not already. PUP-10731

**Puppet feature detection leaves Ruby gems in a bad state**

This release fixes a Ruby gem caching issue that prevented the agent from applying a catalog if a gem was managed using the native package manager, such as yum or apt. PUP-10719

**Fine grained environment timeout issues**

Previously, if the `environment.conf` for an environment was updated and the environment was cleared, puppetserver used old values for per-environment settings. This happened if the environment timed out or if the environment was explicitly cleared using puppetserver’s environment cache REST API. With this fix, if an environment is cleared, Puppet reloads the per-environment settings from the updated `environment.conf`. PUP-10713.

**Setting certname in multiple sections bypasses validation**

Previously, Puppet only validated the certname setting when specified in the main setting, but not if the value was in a non-global setting like agent. As a result, it was possible to set the certname setting to a value containing uppercase letters and prevent the agent from obtaining a certificate the next time it ran. Puppet now validates the certname setting regardless of which setting the value is specified in. PUP-9481

**puppet config set certname accepts upper-case names**

Previously, the `puppet config set` command could set a value that was invalid, causing Puppet to fail the next time it ran or the service was restarted. Now the command validates the value before committing the change to `puppet.conf`. PUP-2173

**3x functions cannot be called from deferred functions in Puppet agent**

This release allows deferred 3.x functions, like `sprintf`, to be called during a Puppet agent run. PUP-10819

**Issue with Puppet creating production folder when multiple environment paths are set**

Previously, the `production` environment folder was automatically created at every Puppet run in the first search path, if it did not already exist. This release Puppet searches all the given paths before creating a new `production` environment folder. PUP-10842

**Puppet 6.19.1**

Released 22 October 2020.

**Resolved issues**

**Masterport not honoured when configured in a section other than main**

Puppet 6.28.0 added the `serverport` config as an alias for `masterport`. This introduced a regression — when `masterport` was set in a section of `puppet.conf` that was not `main`, it ignored `masterport` and set it to the default value. This is now fixed, and both `masterport` and `serverport` are visible in other sections. PUP-10722

**Puppet 6.19.0**

Released 20 October 2020

**New features**

**New server_used report parameter**

This release adds a new report parameter called `server_used`. It behaves like `master_used`. PUP-10672

© 2022 Puppet, Inc., a Perforce company
Removed Puppet::Agent::Locker#running? deprecation warning
The Locker#running? method is still in use. This release removes the deprecation message. PUP-10624

The puppetserver_gem package provider added to core Puppet
Previously, the provider was part of the puppetlabs-puppetserver_gem module, which no longer receive updates. This release moves the puppetserver_gem provider to core Puppet. PUP-10604

Added pip2 package provider when pip3 is the system default
This release adds the pip2 provider to allow you to explicitly use the Python 2 pip. PUP-7526

New environment_timeout_mode setting to clear short-lived Puppet environments from memory
This release adds the environment_timeout_mode setting, which allows you to clear short-lived Puppet environments from memory, while also caching frequently used environments indefinitely. This reduces the Puppet Server memory footprint. For more information, see the release notes summary in the ticket. PUP-8014

Resolved issues

Puppet does not log connection error details when a functional master cannot be located
Previously, errors were only logged at the debug level or at the err level if the no servers were available. Puppet agents now log errors that occur when trying to connect to each server in server_list at the err level. PUP-10664

Performance regression with large hashes using lookup
This release fixes a performance regression in 5.5.x and 6.0x. which affected Hiera lookups in large hashes. PUP-10628

Status endpoint does not use the extra_headers setting
If http_extra_headers is set, Puppet sends the headers for each HTTP request it makes when processing the server_list setting to determine which server to use. PUP-10617

Fact yaml does not quote mac addresses
Previously, when Puppet facts were called with the --render-as yaml option, the macaddress fact was unquoted. This resulted in bad data if other tools parsed the resulted yaml and interpreted the value as a sexagesimal (base 60) integer when the address did not contain chars A-F. This release adds quoting to all yaml strings that contain the : character. PUP-9505

Data type casts cause Puppet Server to retain compiler instances
This issue is now fixed when running under JRuby. PUP-10659

Resource collector overrides cause Puppet Server to retain compiler instances
This issue is now fixed when running under JRuby. PUP-10657

Puppet 6.18.0
Released 25 August 2020

New features

New property and parameter to manage user rights on Windows
You can now manage rights and privileges for local Windows users with the roles property and role_membership parameter. PUP-10547

© 2022 Puppet, Inc., a Perforce company
Curl upgraded to 7.71.1
This release upgrades Curl to 7.71.1 PA-3333

Resolved issues

Add facts and vars back to PAL ScriptCompiler scope
As part of work on PUP-10397, we removed vars and facts from PAL ScriptCompiler which resulted in a breaking change to a public API. This release adds local facts back. PUP-10615

Misuse of URL encoding in file resource HTTP GET
Puppet 6.16.0 introduced a regression when using HTTP file sources, the URL request path or query contained special characters. This release restores the previous behavior. To specify a path or query that contains a space, you need the URL encoded in the manifest. PUP-10603

Puppet lookup loads external facts on the initiating node
Previously, some Puppet default settings required Facter to retrieve data that loaded all external facts and slowed down Puppet. With this release, local external facts are not evaluated when performing a lookup command on a remote node. PUP-10599

The ppt package provider does not mark managed packages as manual
With this release, the apt package provider can now mark packages as manual. PUP-6631

An error is logged during an agent run if Puppet Server does not have at least one <module>/lib dir in its modules directory
If Puppet Server does not have at least one modules directory, the file_metadatas REST API for the plugins, pluginfacts and locales mounts returns file metadata for /etc/puppetlabs/code, instead of returning HTTP 404. PUP-2608

Puppet agent fails to install on a Windows VM if the Windows installer is not used
This release fixes a regression introduced in 6.17.0. You can now install packages when HKLM:\SOFTWARE \Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-18\Components does not exist. PA-3351

Constant corrective changes on SLES 12
This release fixes a regression introduced in Puppet 6.16.0 where the zypper package provider reported corrective changes when the package_version package was installed. PUP-10609

Puppet::Util::Yaml safe_load not loading Time class which leads to compilation error for time serial data
Previously, when an agent run enforced a time-based property of a resource, the next agent run generated an error and lost the ability to report on corrective changes accurately. This occurred when using the {{audit}} metaparameter to monitor file mtime. This is now fixed. PUP-10585

The puppet apply command fails if Puppet types have been generated
Previously, running puppet apply failed if the Puppet generate types command was run before. In this release, puppet apply ignores resource type in the environment resource_types directory. PUP-9602
Puppet run continues despite failed Pluginsync

Previously, Puppet agents ignored pluginsync errors and applied the catalog with incorrect facts and plugin versions, leading to obscure errors or data corruption. This release adds a new setting — ignore_plugin_errors. If set to false, the agent aborts the run if pluginsync fails. The setting defaults to true so the old behavior is preserved. PUP-1763

The puppet_gem provider does not use vendored PKG_CONFIG_PATH when installing native extensions

This release allows you to install gems with native extensions that require PKG_CONFIG_PATH, when using puppet_gem provider, for example the rugged gem PUP-10584

Puppet::Util.default_env method unintentionally removed in 6.17.0

Testing modules using rspec-puppet versions earlier than 2.7.10 resulted in the error undefined local variable or method 'default_env' for Puppet::Util::Module. This is now fixed. PUP-10586

Allow running Puppet commands via AWS Session Manager on Windows 2019

Previously, installing a Puppet agent on EC2 Windows Server 2019 via AWS Session Manager caused an error. This is now fixed. PUP-10600

Puppet 6.17.0

Released 14 July 2020.

New features

Extended trusted_external_command setting

The trusted_external_command setting can now be set to a directory. For each executable file in the directory, the external trusted facts will be stored in the basename key of the trusted['external'] hash. PUP-10528

Updated permissions for files in /cache/state/

This release updates the permission for state.yaml and transactionstore.yaml to remove the group write access. PUP-8922

HTTPS file sources allow additional CA certificates

The releases allows you to configure Puppet agents to trust additional CA certificates when connecting to HTTPS servers when the CA certificate is neither the Puppet CA or in the CA cert bundle of the puppet-agent package. To enable this setting, set ssl_trust_store to a file containing CA cert(s) in PEM format concatenated together, for example, /etc/pki/tls/cert.pem. Note that this setting only applies when Puppet makes an HTTPS request to non-Puppet REST endpoints. PUP-7814

Support for file content from artifactory

Puppet can now retrieve file content from artifactory using an HTTP(S) source parameter. PUP-6114

Support for username:password@domain for HTTP/HTTPS sources

You can now retrieve file content using HTTP basic authentication. The credentials should not appear in debug output or reports. PUP-6916

boost and yaml-cpp dependencies in the SLES 15 puppet-agent package

The boost and yaml-cpp dependencies are now vendored in the SLES 15 agent build. PA-3006
**Added logonaccount and logonpassword**
This release adds the logonaccount and logonpassword fields for managing the logon credentials used by services on Windows. PUP-1289

**Resolved issues**

**Puppet cannot apply a cached catalog containing binary rich data**
Previously, Puppet could not apply a cached catalog if it contained binary or deferred data types, causing it to always fallback to requesting a new catalog. PUP-10572

**File sources with path starting with double slashes are interpreted as a local file**
Puppet incorrectly handled a file resource if the source’s parameter path component had an extra slash, for example, puppet:///modules/mymodule. PUP-10544

**Puppet agents can't download file content from puppetserver via "raw" HTTPS**
If Puppet tries to download file metadata from an HTTP(S) file source and the HEAD request results in "HTTP 403 Forbidden" or "405 Method Not Allowed", it fallback to a GET request with a 0 byte range. PUP-10543

**Resolve Rspec warnings for Ruby 2.7**
You can now run Puppet from source on Ruby 2.7 using bundler without warnings. PUP-10537

**Checksums are not validated when downloading file http(s):// sources**
If a file resource has a desired checksum type and value, but the file downloaded from the remote source does not match, Puppet will raise an error that they mismatch, and will not update the file on the local system. PUP-10368

**HTTP file sources fail for GET-only URIs**
Puppet can now retrieve file content from Amazon AWS and GitHub releases, for example, source => https://github.com/path/to/released/artifact. PUP-6380

**Ensure Windows wide character strings have a wide terminator**
This release improves memory safety when Puppet converts Ruby strings to wide character strings on Windows. PUP-10254

**“HTTP 406 Not Acceptable” error message should contain format names**
This release prints the MIME types, instead of format objects, from the Accept header, in cases where puppetserver cannot generate a catalog in the requested format. PUP-10549

**Puppet Windows package provider fails if there are garbage characters after an embedded NULL**
This release fixes a bug in the Windows package provider when reading package version from registry. This happened if there were garbage characters after the wide NULL terminator. PUP-10536

**Sensitive data type lost when declaring multiple resources using title arrays**
This release fixes support of sensitive data type parameters when declaring multiple resources using title arrays PUP-10511

**Agent runs failing with "Invalid or unsupported charset:ANSI_X3.4-1968"**
Previously, agent runs failed when the locale settings where unrecognised or in an error state because leatherman logging expects a working locale. This release implements a defense for the leatherman logging function. PA-3254
Puppet does not add correct command flags
This release fixes an issue where Puppet would not add the correct command flags when ensuring a user resource with `managehome` enabled. PUP-8897

Puppet agent should ignore `nssm.exe` during an upgrade
Previously, if a puppet-agent upgrade on Windows happened when `nssm.exe` was loaded by EventLog service, certain services were restarted, leading to unreachable machines. This release includes an `nssm` executable for pxp-agent (`nssm-pxp-agent.exe`) and we have remove Windows registry references to the old `nssm.exe`. PA-3263

Puppet should ignore system Ruby settings
Running Puppet using a wrapper script from the puppet-agent package will now ignore the following ruby related environment variables: GEM_HOME, GEM_PATH, DLN_LIBRARY_PATH, RUBYLIB, RUBYLIB_PREFIX, RUBYOPT, RUBYPATH, RUBYSH. PA-3248

Deprecations

The application orchestration features
The application orchestration features are deprecated and will be removed in Puppet 7. As a result, the following will also be deprecated:
- The `{{site}},{(produces)}` or `{{consumes}}` keywords.
- The export and consume metaparameters.
- The environment catalog REST API.

Puppet 6.16.0
Released 3 June 2020.
New features

New puppet help command to generate ronn output
This release adds a `puppet help <app> --ronn` command to generate help text for an application in ronn format. This eliminates the need for `puppet man` which is deprecated and will be removed in Puppet 7. PUP-10502

FileMetadata HTTP terminus to use the new Puppet::HTTP::Client code
Puppet now uses the HTTP client to retrieve file metadata from HTTP(S) sources. PUP-10482

New method for registering runtime implementations
This feature allows caller to register runtime implementations. Currently only HTTP is supported. PUP-10429

Puppet observes `waitforlock` when acquiring the ssl lock
Puppet uses an ssl lockfile to ensure its certificates and keys on disk are not modified by multiple Puppet processes. If Puppet is unable to acquire the ssl lockfile, and the new `Puppet[:waitforlock]` setting is enabled, Puppet will wait that many seconds before retrying the ssl lock. Puppet will wait for `Puppet[:maxwaitforlock]` seconds at most before giving up. PUP-10420

External trusted data is executed for file_content requests
The trusted_external_command is now only run once when external data is requested, instead of every time a request is made. PUP-10292

© 2022 Puppet, Inc., a Perforce company
**Improved error message when a file resource requires a non-existent directory**
This release improves the error message that occurs when trying to manage a file resource whose parent directory does not exist. PUP-4442

**Support on version ranges for the `eq` operator**
Now all the providers that support version ranges will also handle simple versions, making the behaviour consistent between package providers, for example, `1.2.3`. PUP-10496

**New disabled value in `dnfmodule` package provider**
This release adds a disabled value to the ensure field in the `dnf` module provider. PUP-10419

**Added `SemVerRange` support to `zypper` package provider**
This release adds support of `>`, `>=`, `<`, `<=`, `A <=B` ranges for package version specified in `:ensure` for `zypper` provider. PUP-10299

**The Puppet runtime API now accept symbols as names**
This release changes the `Puppet.runtime` method to take symbols instead of strings. PUP-10471

**New Puppet agent options when waiting for lock**
Previously, an agent run would immediately exit if there was an ongoing instance. Two Puppet configuration settings have been added to solve this issue.
• `Waitforlock`: The frequency an agent should run when there is an ongoing Puppet agent instance.
• `Maxwaitforlock`: The maximum amount of time an agent should wait for another agent to finish before starting a new run.
PUP-8051

**New Puppet introduction in the documentation**
An introduction to Puppet to help new users learn the basic concepts and features. See the Puppet introduction.

**Resolved issues**

**Installer switch `service-enable true` reports wrong actions in certain conditions**
Previously, the `puppet resource` command wrongly reporting the enabled state of a `systemd` service when the `ensure` parameter was also specified. This is now fixed. PUP-10479

**OpenSSL errors on AWS FIPS**
Previously, interleaving ruby/leatherman OpenSSL on RehHat7 FIPS led to OpenSSL module errors when the Ruby HTTP client was reused. This is now fixed to adapt `openssl.cnf` and configure FIPS algorithms, by setting `evp_setting` to `fips_mode = true`. PA-3223

**An "unable to set ownership" error with logdest on agents running as root**
Puppet no longer generates an error when using a log file destination. PUP-10407

**Puppet plugin download fails if an agent specified environment does not exist on the agent**
Calling `puppet plugin download --environment <env>` failed if the environment directory did not exist locally. This is now fixed. PUP-10307
The agent incorrectly reports corrective changes when base64 padding is missing from Content-MD5

Previously, when managing file resources with HTTP(S) source parameters, Puppet only parsed the header, causing interoperability problems when the base64 encoded value was not padded. Now Puppet parses the Content-MD5 header according to RFC 2045. PUP-10164

The purge_ssh_keys parameter requires you to set the home directory in the manifest

The release fixes an issue where Puppet required you to manage the home directory when purging SSH keys with purge_ssh_keys. PUP-10506

The pip package provider does not handle pip executable paths with spaces

Paths containing spaces will now be quoted in the pip package providers. PUP-10485

Portage package provider does not work on Gentoo-based distributions

This release fixes an issue where the Gentoo provider was incorrectly selected on Gentoo-family distributions. PUP-10469

Running remove_old_postgresql_versions after upgrade fails

This release fixes an issue in Puppet's behaviour when listing information about a single specific service and all available ones by showing services in a static state. PUP-10437

Zypper provider does not recognize --no-gpg-checks as a global option

This release allows --no-gpg-checks to be passed as an item to install_options property for the zypper package provider. Note that for backward compatibility --no-gpg-check was not removed. PUP-10433

Puppet agent fails to self-restart with configuration changes or updates

Previously, when two agent runs were taking place at the same time, under some circumstances, the Puppet agent could get in a restart_requested state. This is now fixed. PUP-10351

User resource expiry incorrect on SLES 11

On SLES 11, setting a user resource expiry to absent resulted in disabling the user instead. This is now fixed. PUP-7663

The systemd provider does not honor documented enabled states

Previously, enabling indirect services caused an idempotency issue in Puppet. This has been fixed by not allowing users to enable/disable these services and to print a debug log instead. PUP-7163

Deprecations

The puppet module search command

Instead, search for modules on Puppet Forge. PUP-10480

The {find, search, save, destroy, head} methods in the abstract rest terminus

Calling Puppet::Indirection::Rest.{find, search, save, destroy, head} is deprecated as part of an effort to separate the indirector and HTTP requests. If a custom terminus extends the abstract rest terminus, it will implement the networking request itself or call Puppet's HTTP client. PUP-10445

© 2022 Puppet, Inc., a Perforce company
HTTP file content terminus
The HTTP terminus for retrieving file content is deprecated and will be removed in Puppet 7. PUP-10444

Route HttpPool::connection through HTTP client
Calls to Puppet::Network::HttpPool are routed to the new HTTP client, but preserve the existing behavior for that API. The HttpPool class is deprecated and will be removed in Puppet 7. PUP-10289

The Puppet::Network::HTTP::{Connection, NoCachePool, Compression} class
The Puppet::Network::HTTP::Connection class is deprecated and will be removed in Puppet 7. Calls to the Puppet::Network::HttpPool factory methods (such as http_instance) are routed to the new HTTP client in Puppet. PUP-10481

Puppet 6.15.0
Released 30 April 2020.
New features

HTTP client API uses a generic options hash
Instead of using keyword arguments, the HTTP client API uses a generic options hash. This change allows more flexibility with development and ensures there are no problematic version dependencies. PUP-10353

HTTP report processor uses HTTP client
The HTTP report processor uses new Puppet HTTP client when running in puppet apply. Note that it still uses the existing HTTP client when running in puppetserver. PUP-10316

HTTP report processor setting includes system truststore
Previously, when an HTTPS URL was specified, the processor would only trust the Puppet CA when verifying the server SSL certificate. This release adds a Puppet[:report_include_system_store] setting, which defaults to false. If set to true, the report processor trusts CA certificates in the puppet-agent CA bundle, as well as Puppet CA. This change only affects the HTTPS processor when running in the context of puppet apply. PUP-7737

Unified service return values
The return values of the HTTP client service methods now returns the request response, as well as previous return values. PUP-10418

Filebucket rest terminus uses HTTP client code
The filebucket rest terminus now uses the new HTTP client code. All HTTP requests that are routed through the rest terminus will go through HTTP::Client. This includes the logic to find which server to submit requests to. PUP-10385

New metric for Puppet agent run
This release adds a new Startup time metric that accounts for the interval between Puppet agent initialization and the actual run. PUP-10371

New certificate extension OID
This release adds a new certificate extension OID — 1.3.6.1.4.1.34380.1.1.26 — and short name — pp_owner. PUP-10364
**SemVerRange support for yum, apt, pip and gem package providers**

This release adds support for range intersection for the yum, apt, pip and gem package providers. Note that when using version ranges with the yum package provider, there is a limitation that requires you to specify the epoch for the version in the range. See known issues for more information. PUP-10298, PUP-10297, PUP-10296, PUP-10295

**Custom retrieve() method with non-standard return values removed**

This release removes the retrieve() custom method and adjusts tests dependent on the behavior. PUP-9943

**Facts and vars from ScriptCompiler scope removed**

This release removes node specific values from the scope of script compiles, as they did not happen in the context of a node. PUP-10397

**Partitioned reports table**

The PuppetDB reports table is now partitioned by day for better query and garbage collection performance. If you are an open source user, have a large database and do not delete your reports prior to the upgrade, you need to change the start timeout manually. This can take 80-85 minutes per 10GB — note that is an estimate and will be different for every user. PDB-4606

**Ubuntu 20.04 support**

This release adds puppet-agent support for Ubuntu 20.04. PUP-10284

**Resolved issues**

**Upgrade Puppet Server 6.9.2 from Puppet 6.5**

Previously, when querying environments without an environments directory, Puppet Server would report a 406 with an undefined method error. Now it reports an empty environments list. PUP-10405

**The pkgng provider does not parse latest version properly**

The pkgng provider now correctly parses the latest versions for packages. PUP-10382

**The pkgng provider does not support install_options**

The pkgng provider now supports install_options. PUP-10380

**puppet plugin download does not download from locales mount**

This release fixes a regression in 6.14.0 that caused puppet plugin download to not download files from the locales mount in the same way that the agent does. PUP-10374

**Puppet apply recursive copy of directory fails in file resource**

This release fixes a regression in Puppet 6.14.0 that prevented the puppet apply application from recursively copying a directory from a module using a puppet:// URL. PUP-10367

**Puppet agent unable to fetch file from HTTPS source**

This release fixes a regression in 6.14.0 that prevented Puppet agents from retrieving file content from HTTPS sources, when the server's certificate was issued by a CA other than the Puppet CA. PUP-10365

**server_list processed up to four times per agent run**

This release fixes a regression in 6.14.0 that caused Puppet agents to process their server_list up to 4 times per agent run, instead of once per run. This release restores the behavior that existed in 6.13.0 and before. PUP-10363
Can't disable Puppet::SSL::Host deprecation warning when using PuppetDB terminus
Puppet no longer generates a deprecation warning when calling
Puppet::Network::HttpPool.http_instance, which used to happen when using the puppetlabs-puppetdb module to install and manage PuppetDB. PUP-10355

The puppet_gem package provider ensure => absent breaks file resources
The puppet_gem provider can now uninstall a gem during an agent run. PUP-10319

Puppet 6 daemons do not release ssl_lockfile
Previously, daemonized Puppet agents kept the ssl_lockfile locked while waiting for Puppet Server to issue their certificate. This prevented foreground Puppet commands from working. Now the agent releases the ssl_lockfile while it sleeps and reacquires the lock when it wakes up. PUP-10228

Pacman provider does partial upgrades
This release prevents the Pacman package provider from partial upgrades. PUP-10152

Persistent HTTP connection timeouts cannot be changed when using the new HTTP client
The release fixes a regression in 6.14.0 that caused Puppet to cache persistent HTTP connections for 15 seconds, instead of the expected 4 seconds, and it was not possible to change the timeout using the Puppet[:http_keepalive_timeout] setting. PUP-10434

RedHat RHBA-2020:1028 breaks `yum.rb` in yum provider
This release fixes an issue where Red Hat Subscription Manager updates caused ensure => latest to do nothing. PUP-10432

Windows service calls deprecated puppet agent --configprint
This release fixes an issue where Puppet Windows service called deprecated puppet agent --configprint. PUP-10414

Puppet .dmg packages fail on macOS 10.15
In macOS 10.15, the -noidme flag caused .dmg package installation to fail. This release removes the -noidme flag with hdiutil when using pkgdmg package provider. PUP-10408

Gem provider reports changes when using composed requirements
This release fixes idempotency of gem resources with composite requirements, such as range intersections ( >=1.3, <2.2). PUP-10395

Ruby file loaded twice
Previously, the gem package provider’s Ruby file loaded twice and caused warnings on each Puppet run. This release removes an unnecessary file requirement that caused this issue. PUP-10390

Puppet resource service does not return all active services on Ubuntu
This release fixes an inconsistency in Puppet's behaviour when listing information about a specific service. PUP-10378
Custom file ownership for lastrunreport ignored
This release fixes a bug introduced in 6.13.0 where ownership for the lastrunreport file ignored what was defined in puppet.conf. PUP-10376

Custom file mode for lastrunreport ignored
This release fixes a bug introduced in 6.13.0 where the mode for the lastrunreport file was reset and ignored what was defined in puppet.conf. PUP-10325

Latest version ensured for python package is incorrect
Previously, using pip to ensure the latest package available could cause a lack of idempotency, with Puppet identifying the latest version incorrectly. This release improves the comparing and sorting mechanism of versions. PUP-10375

5.5.17 agents fails on interval runs
There were various paths in a Puppet run that did not explicitly return an exit code, leading to a TypeError exception. With this release, the exit argument from run_in_fork is unaffected by these errors. PUP-10366

Facter 4 debug logs do not use Puppet logger
This release fixes an issue where enabling the facterng feature flag stopped the debug logs from being forwarded to Puppet. PUP-10322

Windows user management fails if group contains not resolvable Domain accounts
Previously, if an Active Directory user was added as a member of a local group and then deleted, Puppet did not manage the respective group members. With this release, Puppet will manage the group by showing SID instead of the account name for non-resolvable users. PUP-10303

Windows pidlock access denied
With this release, use SeDebugPrivilege on Windows when opening a lockfile PID to determine whether the process is a Puppet process. PUP-10248

Ruby security update
This version upgrades the Ruby version to 2.5.8 to address security issues:
• CVE-2020-10663: Unsafe Object Creation Vulnerability in JSON
• CVE-2020-10933: Heap exposure vulnerability in the socket library
PA-3176

OpenSSL security update
This version upgrades the OpenSSL version from 1.1.1f to 1.1.1g on non-fips platforms to address CVE-2020-1967, PA-3186

Agent upgrade restarts Windows DHCP Server service
Previously, Windows components (such as DHCP Client) restarted after Puppet agent upgrades. This was a result of nssm.exe being registered as the EventMessageFile, which got loaded and locked into svchost.exe when viewing events raised by pxp-agent. The agent upgrade tried to replace nssm.exe, but failed as it was loaded, and the only way to unload it was to restart. This fix removes the usage of nssm.exe for events and replaces it with EventCreate.exe when generating event messages, so now the EventCreate.exe will get loaded and locked by svchost.exe when viewing Puppet events. This will no longer interfere with the agent upgrade. PA-3175

© 2022 Puppet, Inc., a Perforce company
Puppet 6.14.0
Released 10 March 2020

New features

Hiera-eyaml prints error message when decryption fails
If Puppet fails to decrypt a value stored in hiera-eyaml, include the name of the key where the lookup failed in an error message. PUP-10317

Puppet uses new http client for all REST requests
Prior to this release, the agent only used the http client for certificate related requests. Note that this does not change how Puppet works when running as a library inside Puppet Server. PUP-10260

Send extra headers when requesting a catalog compilation
You can now define custom headers to send with http requests to Puppet infrastructure. Use the :http_extra_headers setting to define these. They should be a comma separated string of key:value pairs. PUP-9566

Puppet HTTP API does not allow callers to trust system cacerts
The HTTP client now accepts a parameter to trust the system certificate store when making HTTPS connections. By default, it only trusts the Puppet PKI. The API is private and may change in a future release. PUP-5069

macOS 10.12, macOS 10.13 and Fedora 28 support removed
This release of the puppet-agent package removes support for macOS 10.12, macOS 10.13 and Fedora 28. PUP-10244, PUP-10245, PUP-10242.

Resolved issues

Disable metrics v1 endpoint and restrict v2 to localhost
To prevent information exposure as a result of CVE-2020-7943, the /metrics/v1 endpoints are disabled by default, and access to the /metrics/v2 endpoints are restricted to localhost. PE-28468

Secondary group handling causing issues in 6.13.0
The members parameter of a group resource was validated by checking if each user exists before the manifest was applied, resulting in a manifest containing both user creation and group members update, which would fail. You can now add new user and update group membership in the same manifest apply. PUP-10320

Client does not reconnect when the server closes a persistent connection
Previously, Puppet failed to reuse a previously cached connection if it was using the new HTTP client, if the previous request was kept alive, or if the server closed the connection between when the client caches it and when the client tries to reuse it. PUP-10347

Replace find_functional_server call with http client
Puppet agents now use the new HTTP client to resolve the first available server in the server_list setting. PUP-10315

Puppet tries to print ssl information for non-ssl connections in debug mode
In debug mode, Puppet now only prints TLS protocols and ciphersuites for HTTPS connections, but not HTTP connections. PUP-10314

© 2022 Puppet, Inc., a Perforce company
Close http client
During the SSL bootstrapping process, Puppet now closes all active HTTP connections before sleeping for "waitforcert" seconds. PUP-10288

When a site has no active connections, it should be removed from the pool.
Reduction in memory usage when the last connection from the HTTP pool is borrowed and not returned. PUP-10286

The puppet apply command does not stream file content
Now the puppet apply command uses the same code path as puppet agent to stream file content from puppet://" and "http(s):// file sources. As a result, the memory footprint is reduced. PUP-8338

The puppet describe --list command requires the description to include a period
The puppet describe --list command now displays summary information for each type and provider, regardless of whether the summary contains a period. PUP-2641

The Puppet agent fails due to a Ruby error
This release fixes an issue with large environment blocks on Windows, which caused a Ruby error. PA-3113

Puppet 6.13.0
Released 18 February 2020

New features
Puppet reads parameter resource for forcelocal set
When managing a user or group resource with a forcelocal set, Puppet reads /etc/passwd or /etc/group for each parameter of the resource. Files are read once per managed resource. PUP-10117

forcelocal parameter available for user and group resources
The forcelocal parameter can be used on AIX for managing local user and group resources, however this option conflicts with the ia_load_module parameter. PUP-10169

Default behavior changed for strict_hostname_checking
Puppet 6.13.0 changes the default behavior for strict_hostname_checking from false to true. It is recommended that Open Source Puppet and Puppet Enterprise users that are not upgrading still set strict_hostname_checking to true to ensure secure behavior. You also must specify the FQDN of the host.

Note: Partial hostname matches are no longer supported, for example node /^foo/

Thank you to Puppet user @Abaddon for reporting this issue.

pe_serverversion optionally added to server_facts
If the file /opt/puppetlabs/server/pe_version exists on an agent then the file will be read and added to server_facts under the name pe_serverversion. PUP-9750

Puppet follows symlink to source directory if versioned_environment_dirs setting is true
If the path to an environment is a symlink and the versioned_environment_dirs setting is true, Puppet follows that symlink to the source directory. PUP-10255
**Improvements to groupadd provider**

The `groupadd` provider now manages members using the `members` resource parameter. PUP-9465

**Update to flavor package parameter**

The `flavor` package parameter is now a property which can be changed after initially set. Supported providers are `dnfmodule` and `openbsd`. PUP-10171

**Agent sleeps then retries connection after 429, 503 message**

When Puppet Server returns a 429 or 503 to an agent after it tries to make an http request, the agent run will sleep and then retry the connection after. Previously, the agent kept the http connection open during the sleep. Now, the connection closes prior to the sleep in order to reduce the resource load on the Puppet Server. PUP-10227

**mark property added as alternative to held value for ensure**

This release adds the `mark` property with Debian and Solaris instead of setting the `ensure` attribute to `held`. The `held` value for `ensure` is deprecated. Allowed values for `mark` are `hold` or `none`, and it defaults to `none`. You can specify `mark` along with `ensure`. If `ensure` is missing, `mark` defaults to `present`. You cannot use the `mark` property together with `purged`, `absent`, or `held` values for `ensure`. PUP-1537

**Puppet docs updated to include OSX Catalina 10.15 (amd64)**

Puppet's public-facing documentation now includes information on this platform. PUP-10111

**Resolved issues**

**Puppet incorrectly parsed XML plist data**

Previously, Puppet tried to parse an XML property list file as ASCII instead of UTF-8. This fix resolves the issue. PUP-10241

**Missing ppAuthCertExt added to custom_extensions**

The `custom_extension` method allows certificate extensions to access the manifest. Although, ppAuthCertExt was added it never updated to be included. This fix adds the missing ppAuthCertExt OID and updates docs to reference it. PUP-10234

**Functions slowed when called with chained syntax**

The Puppet language supports calling functions using prefix or chained syntax, for example: `each($var)` and `$var.each`, respectively. Chained syntax should result in the same behaviour as prefixed syntax. However, chained syntax preform slower. This fix addresses the gap. Now, chained syntax is much faster, especially with a large hash, such as `$facts` from the node. PUP-10113

**Array settings printed poor default values and error reporting**

Puppet now prints a clear error message if the `disable_warnings` setting is specified incorrectly: "Cannot disable unrecognized warning types foo. Valid values are `deprecations`, `undefined_variables`, `undefined_resources". PUP-2820

**puppet facts upload face only tried first primary server in server_list**

`puppet facts upload` now tries each server entry in the `server_list` setting, similar to how `puppet agent` does. Previously, it only tried the first entry in the `server_list` setting. PUP-8900
**PObjectTypeExtension doesn't implement implementation_class**

Objects that inherit from the `{{PObjectTypeExtension}}` base type—for example `{{Puppet::Datatypes::Error}}` objects—raise errors when serialized to pcore due to missing methods. The missing methods have been implemented and the types are now able to be serialized to pcore. PUP-10221

**Puppet incorrectly detected stale pidfile**

When an agent is incorrectly terminated, it occasionally detected a stale pidfile. This fix orders a pidfile lock removal when an agent is started as a LightWeight Process (LWP) and is incorrectly terminated on POSIX operating systems. PUP-10218

**Yum provider doesn't allow array of strings**

The Yum package provider now accepts an array of strings to be specified for the `install_options` parameter in addition to the previous implementation which only accepted an array of hashes. Fix provided by community contributor Corey Osman. PUP-10177

**Fixed syntax error in pxp-agent init script**

The pxp-agent init script no longer errors if there are multiple instances of the service running. PUP-889

**puppet resource now returns the name of the provider**

The `puppet resource <type>` command returns resources from all suitable providers. Previously, the name of the provider was not included in the output. However, the result was ambiguous for resource types like `package` which support multiple providers at the same time, for example `yum` and `gem`. The `puppet resource` command now returns the name of the provider so that resources are uniquely identified. PUP-3721

**Couldn't set, change password of a user resource in macOS 10.15**

Due to a change in user management in Apple's macOS 10.15, Puppet could no longer manage user passwords on that version. Puppet now uses the `dsimport` utility to manage passwords on macOS 10.15 or higher. PUP-10246

**Mailalias performs a change when value has a comma**

When a mailalias resources is used, Puppet refreshes the alias without prompting, changing the value. To mitigate this, do not use commas. Mailalias records field separators when surrounded by double quotes. Example: "...project" is okay, but "...project," will cause an issue). PUP-10287

**Puppet agent cannot run as administrator if first PA run is done as system**

If the first Puppet agent run was done under SYSTEM account, following runs done by administrator users failed to send the report to primary server. This was due to insufficient file permissions caused by the implementation of `Puppet::Util.replace_file`. Puppet now uses the new `Puppet::FileSystem.replace_file` method which correctly handles files permissions on Windows. PUP-9719

**Puppet 6.12.0**

Released 14 January 2020

**New features**

**Update to compilation warnings**

This release includes improvements to the evaluator, meaning some compilation warnings now take less time to compute. PUP-10213
**Improvements to some manifest function performance**

Performance of manifests that use the `PuppetStack.top_of_stack` function have been greatly improved. This includes manifests that use the puppetlabs-stdlib `deprecation` function or the pseudo keywords `break`, `return`, and `next`. [PUP-10170](#)

**Merge dependency warnings**

If a class has a failed dependency every resource in that class generates a `notice` level message about the dependency failure and a `warning` level message about skipping the resource. At large-node or large-code scale, one dependency failure can create an overwhelming number of warnings. To collapse all messages caused by one class dependency failure into one message associated with the class, set `merge_dependency_warnings=true`. [PUP-10017](#)

**OpenSSL 1.0.2 updated to 1.1.1d**

OpenSSL 1.0.2 reached end of life on 31 Dec 2019. The OpenSSL version used in Puppet 5.5.x has been upgraded to OpenSSL 1.1.1d. [PUP-3029](#)

**Deprecation warning for Windows Server 2008 and 2008 R2**

On January 14, 2020 support for Windows Server 2008 and 2008 R2 ends. [PA-3018](#)

**puppet module install uses Puppet’s HTTP client**

The `puppet module install` command now uses Puppet’s HTTP client to connect to the Forge. It observes the `Puppet[:sourceaddress]` setting, useful when running the install command on a `puppetserver` with multiple network interfaces. [PUP-10041](#)

**Systemd is the default service provider for Cumulus 4**

This release adds systemd as the default service provider for Cumulus 4. PR contributed by Brian Rak [PUP-10162](#)

**Resolved issues**

**Introduces puppet_trace and bug fix in trace**

This release fixes a bug where stacktraces from errors no longer had the Ruby stack frames interleaved with the Puppet stack frames when using `trace`. This release also introduces a new setting, `puppet_trace`, which prints the Puppet stack without the Ruby frames interleaved. If the `trace` setting is enabled, it overrides the value of `puppet_trace`. [PUP-10150](#)

**Puppet loaded types and providers during environment convergence**

Previously, Puppet agents could fail to apply a catalog if the agent switched environments based on node classification and if there were different versions of a module in those environments. As a result of this fix, an agent loads types and providers only once, convergences to its server-assigned environment quickly, and updates its cached catalog after the environment converges. [PUP-10160](#)

**Premature loading of module-provided facts under --p**

Previously, when running `facter --p` on Windows with custom facts or module facts that required additional files that were not present in `$LOAD_PATH`, an error would occur. Now, `$LOAD_PATH` are set without using Facter, getting it directly from the system environment. [PUP-10136](#)

**Debian Puppet-agent package lacked SELinux Ruby library**

Ruby SELinux libraries are now also provided for Debian and Ubuntu platforms. [PA-2985](#)

© 2022 Puppet, Inc., a Perforce company
**puppetdb_query didn’t respond to hostprivkey & hostcert settings**
Previously, an override of the client certificate and corresponding private key in Puppet settings using the `hostcert` and `hostprivkey` was not possible. This fix restores that capability. **PUP-10165**

**Package provider removed colon from package name**
This fix corrects the implementation of a colon (`:`) as a version slot separator for Gentoo Linux. **PUP-10124**

**When checking binary file changes, puppet agent -t, puppet apply --show_diff occasionally generated an error**
Previously, `puppet agent -t` or `puppet apply --show_diff` could generate an error when trying to display the changes it made to a binary file. Puppet now detects this case and prints a generic message stating that the binary files differ. **PUP-10097**

**Windows confused domain and local accounts**
Puppet no longer checks for domain users or groups when managing local resources on Windows. This fix addresses a local user management issue occurring when an Active Directory account existed with the same name as the local user. **PUP-10057**

**Fixed HP-UX usermod syntax**
The HP-UX provider forced command line arguments to `usermod` to be in a specific order. This is now fixed. **PUP-9391**

**server_list setting not observed when making CA requests**
The CA service now observes the ServerList resolver when attempting to resolve routes. Previously, SRV Records or the `server/ca_server` setting found this information. **PUP-10157**

**Could not create local user when username already exists in Windows domain**
As of this release, Puppet ignores case for the domain/computer name comparison when checking for local accounts on domain-joined Windows machines. **PUP-10219**

**Template rendering updated to enhance ease of use**
Secret values resolved by deferred functions can now be used directly with tools like Vault. This fix makes it easier to render templates on the agent. **PUP-10139**

**Password did not update when account disabled for Win 2012 R2**
Puppet can now change the password for a non-active account on Windows. **PUP-10021**

**Regression: Puppet 6.11 can’t manage SELinux on Centos/Redhat 8**
This release fixes an issue with management of SELinux using Puppet on RedHat and Debian based systems. **PA-3067**

**Puppet 6.11.1**
Released 20 November 2019

**Resolved Issues**

**Fixed systemd daemon-reload issue**
This release fixes a regression where the `daemon-reload` method returned all properties instead of the value for a specific `NeedDaemonReload` property. Contributed by alexjfisher. **PUP-10149**

© 2022 Puppet, Inc., a Perforce company
Puppet 6.11.0
Released 19 November 2019

New features

SSL state machine and application now use the Puppet HTTP client
This release modifies `puppet ssl`, `puppet device`, and `puppet agent` to use the Puppet HTTP client to bootstrap SSL certificates on the node. PUP-10040

Virtual package support for apt and dpkg providers
To allow virtual packages, ensure that the `packages` attribute `allow_virtual` is set to `true` in your Puppet resource file. PUP-10023

Package support for DNF modules
Puppet now supports managing DNF modules, which are groups of packages that represent an application, a language runtime, or any logical group.
Modules can be available in multiple streams, usually representing a major version of the software they include. Profiles are package subsets representing a specific use case of the module (these are handled by the flavor parameter of the package type).
Due to the significant difference between a package and a module, `dnfmodule` is an opt-in provider and should be explicitly specified in the manifest. PUP-9978

Boolean support for `selboolean`
The `selboolean` value property now accepts `true` and `false` values in addition to `on` and `off`. PUP-9963

Windows improvements
This release removes a dependency on .bat files when running Puppet as a service on Windows. PUP-9940

Resubmit facts at the end of an agent's run
Puppet submits facts when requesting a catalog, but if the agent modifies the system while applying the catalog, then the facts in PuppetDB won't be refreshed until the agent runs again, which may be 30 minutes depending on how `runinterval` is configured. This feature makes it possible to submit facts again at the end of the agent's run, after the catalog has been applied. To enable this feature, set `resubmit_facts=true` in the agent's `puppet.conf`. Resubmitting facts doubles the fact submission load on PuppetDB, since each agent will submit facts twice per run. This feature is disabled by default. PUP-5934

Experimental feature: catalog compilation with external trusted data from third parties
Use `trusted_external_command` to add trusted facts from an external source during catalog compilation.
The value for `trusted_external_command` is a path to an executable command that can produce external trusted facts. The command must:
- Take the name of a node as a command-line argument.
- Return a JSON hash with the external trusted facts for this node.
- Exit with a non-zero exit code for unknown or invalid nodes.
PUP-9994

Hardened Linux Executable and Linkable Format (ELF) binaries
Linux ELF binaries and shared objects are now compiled with Position Independent Executables (PIE) support, stack canary, and Relocation Read-Only (RELRO). PA-2986
Resolved issues

Certificate requests sometimes raised confusing error messages

Previously, when Puppet encountered a connection error, it would create a new exception with additional contextual information around what was causing the error. However, this new exception could cause an additional "Wrong number of arguments" error. Puppet now raises the original error and logs it with any additional contextual information. **PUP-10121**

The no_proxy setting ignored FQDN suffixes unless they had a leading wildcard or period

Previously, Puppet would only bypass a proxy if no_proxy had a leading wildcard or period. For example, *.example.com or .example.com. Puppet now bypasses the HTTP proxy if the no_proxy environment variable or puppet setting is a suffix of the destination server FQDN. **PUP-10106**

Puppet device runs would fail when specifying csr_attributes.yaml

The csr_attributes.yaml file can now be specified when requesting a certificate signing request for a device using puppet device --target devicename. **PUP-10104**

On certain Linux distributions, Puppet could incorrectly handle the service setting

On Linux distributions that do not have a default service provider, Puppet sometimes defaulted to using the defpath before verifying that runit was suitable. **PUP-10102**

Error messages for exec commands with paths that could not be resolved included sensitive data passed to the command

If an exec resource's command is not executable or cannot be resolved into a fully qualified path, Puppet now only prints the command, and not the potentially sensitive arguments passed to the command. Puppet also redacts the output of sensitive commands when the logoutput parameter is set to true, or the parameter is on_failure (the default), and the command fails. **PUP-10100**

Puppet wouldn't install dpkg sub-packages when ensure was set to held

Puppet now correctly installs dpkg sub-packages and sets them to held if ensure is set to held. **PUP-10059**

skip_tags was missing help documentation

Added help documentation for skip_tags. Contributed by cvquesty. **PUP-10026**

The deep_merge Ruby gem was vendored by Puppet

This release adds a runtime gem dependency for the deep_merge gem and ensures that the gem is no longer vendored. **PUP-10014**

Puppet couldn't manage pip resources if the pip command was in a directory containing spaces

Puppet can now manage pip resources in directories containing spaces, such as C:\Program Files \Python27 on Windows. **PUP-9647**

User resource did not respect forcelocal for the comment parameter

Prior to this release, a user resource configured with forcelocal would still try to sync the comment parameter with the external directory services. **PUP-9562**
Pluginsynced module API improvements

If a module used a legacy Ruby (Puppet version 3) function, and it was pluginsynced (via puppet agent -t), the function could not be loaded when using Puppet as a library. Puppet 3 API functions are now available from the cached lib directory. If a function exists in both version 3 and version 4, Puppet uses the version 4 implementation. PUP-9509

Improved handling of pip version detection during catalog compilation

Previously, the pip provider failed if pip --version did not emit the version on the first line of output. PUP-8986

The pip package provider did not use install_options when ensure was set to latest

Puppet now uses install_options whenever pip install is executed. PUP-8949

The systemd service provider failed on services whose names started with a dash

The systemd service provider can now manage services whose names start with a dash. Contributed by j-collier. PUP-7218

Improvements to working directory handling

Previously, if the cwd parameter was not specified, puppet would change its working directory to the current working directory, which was redundant and could fail if the current working directory was not accessible. Now, exec resources only change the current working directory if the cwd parameter is specified in a manifest. PUP-5915

Puppet was unable to manage passwords for disabled Windows users

Puppet can now manage passwords for disabled Windows users. PUP-10021

Puppet 6.10.1
Released 15 October 2019

New features

Module installation performance with minitar improved

Installation time on larger modules has been improved. Previously, on platforms that had the minitar gem installed, minitar would fsync every directory and file, causing long extraction times during module installation. Puppet now uses minitar 0.9, with the fsync option turned off by default. PUP-10013

Automatic (delayed start) is now an option for Windows services

Puppet can now set Windows service startup type to Auto-Start (Delayed). To set a service to use this setting, set the enable parameter of the service resource to "delayed". PUP-6382

Resolved issues

Module repository credentials redacted during installation

When you install a module, if the module repository URL includes credentials, Puppet now redacts the credentials in the command line output. PUP-9787

Sensitive values redacted in notify resource messages

Prior to this release, the notify resource leaked data if the message was a sensitive datatype with a raw value, not encapsulated in quotes. Now sensitive values are redacted when they are interpolated in a notify resource's message. PUP-9295
Ruby security update
This version upgrades the Ruby version to 2.5.7 to address security issues:

• RDoc vulnerabilities
• CVE-2019-16255: A code injection vulnerability of Shell#[] and Shell#test
• CVE-2019-16201: Regular Expression Denial of Service vulnerability of WEBrick’s Digest access authentication
• CVE-2019-16254: HTTP response splitting in WEBrick
• CVE-2019-15845: A NUL injection vulnerability of File.fnmatch and File.fnmatch?

Curl security update
This version includes a security update to curl 7.66.0 to address CVE-2019-5481 and CVE-2019-5482.

OpenSSL security update
This version upgrades OpenSSL to 1.1.1d to address CVE-2019-1547, CVE-2019-1549 and CVE-2019-1563. For more details, see the OpenSSL Security Advisory.

Puppet 6.10.0
Released 1 October 2019
Resolved issues

Puppet tried to install packages that were already installed
Previously, Puppet incorrectly parsed the output of `pip freeze` when it reported package versions using the arbitrary equality operator, `===`. As a result, Puppet treated the package as not installed and tried to reinstall it during every Puppet run. PUP-10015

Query parameters for HTTP and HTTPS file resources are preserved
When retrieving metadata and content for HTTP or HTTPS file resources, Puppet now preserves query parameters. Previously, Puppet requested only the path element of the URI and skipped the query parameters. PUP-9109

YAML output with special characters was not valid
Previously, the `puppet resource --to_yaml` and `puppet device --to_yaml` commands did not generate valid YAML if the output contained special characters such as a single quote. PUP-7808

Puppet 6.9.0
Released 17 September 2019
New features

serverip6 fact added
This release adds the `serverip6` fact, which returns the server's IPv6 address. If Puppet cannot find either `serverip` or `serverip6` facts, it returns a warning. PUP-5109

Agent startup logged at debug level in daemon mode
When running in daemon mode, Puppet logs the configuration used on agent startup at the debug level. The log is sent to the output specified by the `--logdest` option. Configuration is reloaded and also logged on SIGHUP. PUP-9754

puppet plugin information was not included in help
The `puppet help` command did not include help information for `puppet plugin`. Now the `plugin` command is included in `puppet help` output. PUP-9959

© 2022 Puppet, Inc., a Perforce company
Resolved issues

**systemd could not run as non-root**

This release fixes a regression introduced in Puppet 6.8.0, where the `systemd` provider was prevented from working when running as non-root. For example, the command `puppet resource service` did not include `systemd` services. PUP-10016

Puppet

**puppet module uninstall now works in FIPS mode**

Previously, the `puppet module uninstall` command failed if you ran it in FIPS mode. With the release, `puppet module uninstall` now works in FIPS mode, provided you specify either the `--ignore_changes` or `--force` arguments. If you do not specify one of these arguments, then uninstall fails, so that local changes to the module are not lost. PUP-9991

**no_proxy setting correctly bypasses proxy even when HTTP_PROXY environment is set**

Fixes a bug where Puppet attempted to use a proxy specified in the `HTTP_PROXY` environment variable, even though the `no_proxy` setting was set to bypass the proxy. PUP-9990

**exec conditionals respect sensitive types**

The `exec` type's `onlyif` and `unless` checks now return redacted output if it is marked sensitive. PUP-9956

**install_only parameter added for yum, dnf providers**

This release adds a new parameter, `install_only`, for `yum` and `dnf` package providers. Set this parameter for packages that should only be installed, but never updated, such as kernels. PUP-9605

**Plug-in download speed improved**

The `puppet plugin download` command now reuses HTTPS connections. This significantly speeds up the download process. PUP-8662

**Puppet no longer ignores truncated file downloads caused by a Ruby issue**

Prior to this release, Puppet silently ignored truncated file downloads, such as when using a file resource whose source parameter contained `puppet://`, `http://`, or `https://` URL. This issue was caused by a Ruby issue and is fixed in this release. PA-2849

**Puppet 6.8.1**

Released 28 August 2019

**Resolved issues**

**Virtual package support removed for apt and dpkg providers**

This release removes the virtual package support added for the `dpkg` and `apt` providers in the previous release. Puppet 6.8.0 added virtual package support to these providers with an `allow_virtual` setting that defaulted to `true`. This caused issues because virtual packages are not enabled by default on `apt` and `dpkg`. PUP-9974

**Fixed command execution with certain sensitive non-string arguments**

Puppet could not execute commands with non-string arguments if the command was also marked as sensitive and the arguments were passed as an Array to the `execute` method. The issue is fixed for all providers, including the `useradd` provider. PUP-9973
Puppet 6.8.0
Released 21 August 2019

New features

**puppet module install command prefers SHA-256 for verifying tarball**

The `puppet module install` command now prefers SHA-256 when verifying the downloaded module tarball, but falls back to MD5 if necessary. [PUP-9909]

**puppet-agent packages for Debian 10 added**

A `puppet-agent` package for Debian is now available.

**Virtual packages support for `apt`, `dpkg`**

This release allows `apt` and `dpkg` providers to install virtual packages. By default, the `allow_virtual` setting defaults to `true`, which enables the ability to install virtual packages.

If a package is defined as virtual, by default Puppet searches for a real package on that system. For example, for a virtual package named `rubygems`:

```ruby
package { 'rubygems': ensure => 'latest' }
```

Puppet finds and installs the real package `ruby`. You can also uninstall a virtual package this way, by setting the package's `ensure` attribute to `absent`:

```ruby
package {'rubygems':
  ensure => 'absent',
  allow_virtual => false
}
```

By default, this removes the `ruby` package. To disable virtual packages, set `allow_virtual` to `false`.

This feature also address the Ubuntu reporting issue in [MODULES-2047]. [PUP-9909]

**New no_proxy setting available in puppet.conf**

You can now specify `no_proxy` as a Puppet setting, consistent with other `http_proxy_*` Puppet settings. The `NO_PROXY` environment variable takes precedence over the `no_proxy` Puppet setting. [PUP-9316]

**bolt-shim module no longer required to run scripts over ppx-agent**

Previously, to run a script over a `ppx-agent`, you had to first install the `bolt_shim` module. This fix allows `ppx-agent` to respond to requests to execute a script run action without the module. [PUP-869]

**Resolved issues**

**Agents now connect directly if target host is set to NO_PROXY**

If the agent is configured to use an HTTP proxy, and it attempts to connect to a host that matches an entry in the `NO_PROXY` environment variable, then Puppet connects directly to the host instead of using the proxy. This feature was originally introduced in Puppet 4.2, but it did not work. [PUP-9942]

**Windows components restarted on agent upgrades**

During `puppet-agent` upgrades, Windows Network Components were sometimes restarted. This occurred when the upgrade process tried to modify a DLL file that was already loaded. This release fixes the issue, and Windows components no longer restart during upgrades. [PA-2701]
Conflict between puppet-agent and puppetserver packages resolved

Puppet might fail to load under JRuby 9.1, if puppet-agent 6.4 or greater and puppetserver 6.0.x were installed on the same host. PUP-9927

Services incorrectly reported as absent after change

When Puppet runs a resource service that changes the state of the service, the returned status would always fail, causing the ensure status of the service to be absent. Now Puppet correctly returns the service status when a change of the service is requested. PUP-9908

Passing code to the parser failed on STDIN

Resolved an issue where puppet parser validate failed with an error when a valid manifest was passed in through the STDIN pipe. PUP-9816

Puppet selects systemd

Puppet selects systemd as the default service provider on Debian 10, "buster". PUP-9752

Puppet repeated file mode changes on some Windows files

When puppet apply changes the mode on a file inside a protected Windows directory, Puppet changes the mode on every run, even if the desired mode matches the current mode. This fix improves the analysis of the file mode, so that if the desired mode matches the current mode, Puppet does not set the mode. PUP-9722

Passwords for the user type were output in Debug logging

User providers will not output exec command lines with passwords during debugging level logging. Instead, the passwords are now marked [redacted]. PUP-9585

Device certificates could be lost in backup and restore

Prior to this change, Puppet device certificates were stored under the cache directory. When the proxy agent for a device is the primary server, and the user executes a backup and restore, either manually or automated as in OpsWorks, the certificate and keys were lost, requiring certificate regeneration for those devices.

This fix moves the device certificate directory to a persistent path, leaving the old path as a symbolic link for compatibility purposes.

If you are using the device_manager module to manage your devices, upgrade to 3.0.1 or later to avoid deleting certificates from their new location. PUP-8736

systemd provider confined to init system matching systemd

This change allows the systemd provider to work in a chrooted environment. PUP-7312

HTTP connections did not support authenticating proxies

Agents could not connect through an authenticating HTTP proxy when making REST requests to Puppet infrastructure, such as when requesting a catalog. Now agents will observe the http_proxy_user and http_proxy_password settings or HTTP_PROXY_USER/PASSWORD environment variables when making those requests. PUP-4470

Puppet does not use proxy to connect to localhost

If an HTTP proxy is configured either in Puppet settings or the HTTP_PROXY_* environment variables, then Puppet does not use the proxy when connecting to localhost or 127.0.0.1. This behavior can be modified by changing the no_proxy setting in puppet.conf or the NO_PROXY environment variable. PUP-2172
Puppet was unable to appropriately manage SELinux file contexts

Previously, Puppet was unable to effectively manage the SELinux file context without restarting the Puppet service. This issue was due to an infinitely cached default context for each file. Now Puppet clears this cache after each catalog apply. PUP-2169

Deprecations

SSL-related API classes deprecated

Various classes are marked as deprecated in API documentation, and will be removed in a future major version of Puppet. Constructing an instance of Puppet::SSL::Host, attempting to get or set Puppet[:ssl_server_ca_auth], or attempting to set Puppet[:ssl_client_ca_auth] now generates a deprecation warning. PUP-9513

The following classes or methods are deprecated and will be removed in a future version of Puppet:

- Puppet::Network::HttpPool.http_instance
- Puppet::Network::HttpPool.http_ssl_instance
- Puppet::SSL::Host

Indirected Types:

- Puppet::SSL::Base
- Puppet::SSL::Key
- Puppet::SSL::Certificate
- Puppet::SSL::CertificateRequest (just the indirector related stuff, the rest is ok)

Termini:

- Puppet::Indirector::SslFile
- Puppet::SSL::Key::*
- Puppet::SSL::Certificate::*
- Puppet::SSL::CertificateRequest::*

Validators:

- Puppet::SSL::Validator*
- Puppet::SSL::Validator::*
- Puppet::SSL::VerifierAdapter

Settings:

- ssl_client_ca_auth
- ssl_server_ca_auth

Puppet 6.7.2
Released 26 July 2019

Resolved issues

Catalog application failed with puppetlabs-ciscopuppet module

Catalog application failed when using a custom resource type whose automatic relationship method, such as autorequire or autobefore, returned an instance of Puppet::Type instead of a resource name. This occurred in particular when using the puppetlabs-ciscopuppet module. This issue was caused by a regression in Puppet 6.7.0. PUP-9926
Use of the mailalias resource type returned errors

This release fixes an issue affecting the mailalias resource type. Puppet was unable to correctly parse files, resulting in errors such as:

```
Could not prefetch mailalias provider 'aliases': Could not parse line "mailer-daemon: postmaster" (file: /etc/aliases, line: 12)
```

This issue was caused by a regression introduced in Puppet 6.7.0. PUP-9914

Puppet 6.7.1
This version of Puppet was never released.

Puppet 6.7.0
Released 23 July 2019
New features

New ca_fingerprint setting verifies the CA bundle download against a fingerprint

This release adds a ca_fingerprint setting to specify that newly provisioned agents should verify the CA certificate when it is initially downloaded. This provides a way to securely bootstrap new agents. To enable this verification, set ca_fingerprint to the SHA256 digest of the CA certificate, which can be calculated on the Puppet Server. To calculate this SHA, run the following command on the primary server, or on the CA server if you have specified a server other than the primary server:

```
openssl dgst -sha256 -r /etc/puppetlabs/puppet/ssl/certs/ca.pem | cut -f1 -d' '
```

PUP-9638

Resolved issues

Some commands could not be found

Some Puppet commands, such as puppet-infra, might not be found in the system PATH. This fix ensures that the relevant directory, opt/puppetlabs/bin, is available in the PATH. PA-2750

Custom MSI actions are logged

Custom MSI actions did not correctly log STDERR to the MSI log. PA-2691

Puppet 6.6.0
Released 1 July 2019
New features

--evaltrace option now shows total number of resources to be evaluated

The --evaltrace argument, used with the puppet agent and puppet apply commands, now returns the number of resources evaluated and the total number of resources to be evaluated. This option is useful for showing which resource in the catalog is being evaluated and the progress through the current catalog. PUP-9465

Resolved issues

Provider autoloading issue fixed

In the 6.5.0 release, Puppet was unable to autoloading providers on some systems. This issue has been fixed. PUP-9794
**pip provider error fixed**

In the 6.5.0 release, Puppet encountered an undefined method error when running the pip provider. This issue has been fixed. PUP-9790

**puppet agent --fingerprint returns the CSR hash**

When you run the puppet agent --fingerprint command, if the agent doesn't have a client cert yet, the Puppet returns the SHA256 digest of the certificate request (CSR). This functionality was broken as of Puppet 6.4.0, and is now fixed. PUP-9720

**Recurring Puppet runs exited on some SSL bootstrap errors**

Recurring Puppet runs wait a specified amount of time while bootstrapping the SSL system, and then retry if an error is encountered. This behavior was broken as of Puppet 6.4.0, and this release restores the behavior.

The wait interval is controlled by the waitforce setting. One-time Puppet runs such as puppet agent --test or puppet agent --onetime do not retry, and instead exit when the first error occurs. PUP-9717

**Lockfile retained old PID, causing agent failure**

This release fixes an issue where if a Puppet run is killed, the lockfile containing the PID that was being used for the process remains. If another process subsequently starts and uses this PID, the agent fails. Puppet now checks that the PID belongs to Puppet so it can lock the PID correctly. This fix works for Puppet even if you run it as a gem. PUP-9691

**Puppet now registers OIDs in the SSL application**

SSL requests might sometimes return errors because Puppet was not registering OIDs in the SSL application. This is now fixed. PUP-9746

**Puppet 6.5.0**

Released 19 June 2019

**New features**

**Use the staging_location parameter to customize the temporary location for new files**

You can now use the staging_location parameter to render a file in a different location before you validate it with the validate_cms parameter. PUP-9389

**Use puppet catalog compile to compile catalogs**

The puppet catalog compile action works in the same way puppet server --compile worked before it was removed in Puppet 6.0.0. You must run the command on the puppetserver with access to your environments, modules, manifests, and Hiera data. PUP-9055

**Create a Regexp with all special characters escaped**

Create a Regexp that escapes all regexp special characters by adding a Boolean true as a second argument in a call to new. For example, Regexp("./", true) or Regexp.new("./", true). The default value is false. PUP-9554

**Package providers now support an alternative gem command**

Previously, package providers relied on $PATH for the default gem. This release adds a targetable feature to the package type, allowing package providers to implement a command attribute. The gem and pip providers now implement that attribute. This feature allows Puppet to manage packages in software collections, such as Red Hat Software Collections. PUP-6488

© 2022 Puppet, Inc., a Perforce company
For example:

```yaml
package { 'colorize':
  name  => 'colorize',
  provider => gem,
}

package { 'colorize-opt':
  name  => 'colorize',
  provider => gem,
  command  => '/opt/ruby/bin/gem',
}
```

Manage multiple Python installations with `pip` and `pip3` package providers

You can now add a custom path to your `pip` and `pip3` providers using a package command. If you have multiple Python installations, this allows you to point to a specific installation. PUP-1082

Specify the maximum amount of time an agent should wait for its certificate

By default, Puppet agents attempt to download their signed certificate indefinitely. This release adds a `maxwaitforcert` setting, which specifies the maximum amount of time an agent should wait for its certificate. Acceptable values are `unlimited` (the current behavior), or a duration such as `10m`, or `1h`. If you specify a duration, the agent waits the full amount of time and, if the certificate is not downloaded, exits with an error. PUP-3237

Elliptic-curve cryptography (ECC) key support

Configure your agent to use elliptic curve private keys using the `key_type=ec` setting. By default, Puppet uses the `prime256v1` elliptic curve, but you can specify an alternate curve using the `named_curve` setting if the curve is supported by Ruby and OpenSSL. See `OpenSSL::PKey::EC.builtin_curves` for a list of supported curves. PUP-2606

**Note:** Puppet ignores the `key_type` and `named_curve` settings if the agent already has a private key. These settings only control the type of private key that the agent generates. The settings do not affect which curve is selected in the TLS protocol.

Specify a refresh interval for certificate revocation lists (CRLs)

Use the `crl_refresh_interval` setting to specify a refresh interval for CRLs. If specified as a duration, such as `8h`, or `7d`, the agent refreshes its CRL on its first run after the specified duration has elapsed. If the agent downloads a new CRL, it uses the new CRL for all subsequent network requests. If the refresh request fails or if the CRL is unchanged on the certificate authority (CA), the agent run continues using the local CRL. PUP-2310

**Note:** Always set the duration to be greater than the `runinterval`. Setting `runinterval` to an equal or lesser value than the duration causes Puppet to refresh the CRL on every agent run.

Improved `server_list` output and error messaging

Previously, using `config print` to view your `server_list` would output a nested array that was difficult to read. Using `config print` now outputs the text in the same human-readable format as its entry in `puppet.conf`. Puppet uses the same human-readable output for errors you receive from being unable to connect to a server in `server_list`. PUP-9495
**Improved JSON output support for validation errors**

The `puppet parser validate` subcommand now supports a `--render-as=json` option to output validate errors in a machine-readable JSON format. Additionally, if you use the command with multiple files, Puppet continues to validate additional files when it finds a parse error, instead of halting immediately on the first error encountered. PUP-8984

*Note:* `puppet parser validate` returns a maximum of one parse error per file.

**Fedora 30 support**

This release adds `puppet-agent` support for Fedora 30. PA-2675

**Resolved issues**

**puppet device failed to manage multiple devices**

The `puppet device` command would not manage multiple network devices in a single run. This was a regression introduced in Puppet 6.0.5. PUP-9587

**Security update to curl**

This release includes an update to curl to address security issues. See [https://curl.haxx.se/docs/CVE-2019-5435.html](https://curl.haxx.se/docs/CVE-2019-5435.html) and [https://curl.haxx.se/docs/CVE-2019-5436.html](https://curl.haxx.se/docs/CVE-2019-5436.html) for information about the CVEs. PA-2689

**Amazon platforms now use yum as the default provider**

Prior to this release, Amazon platforms did not have a default provider set. This resulted in Puppet trying to use the `gem` provider to install Amazon packages. PUP-9724

**On Windows, Puppet no longer applies corrective changes to the administrator password on every Puppet run**

Puppet now applies corrective changes to the administrator password only on the first run. PUP-9688

**Improved syntax error feedback for legacy Ruby functions**

If you loaded a legacy Ruby function with syntax errors, you'd get an error saying that your function "does not seem to be a Puppet 3x API function." Puppet now alerts you to syntax errors. PUP-9643

**Password protected private key support for agent-only nodes**

If a private key password file (`Puppet[:passfile]`) exists and the agent doesn't yet have a private key, the agent generates a key and uses the contents of the `passfile` to encrypt the key on disk using AES-128-CBC. If the agent already has an unencrypted private key, no change occurs. PUP-9466

*Note:* Puppet Server does not support password protected private keys. You can enable password protected private keys on agent-only nodes.

**Temporary files created from validate_cmd use the same permissions as the file resource**

Prior to this release, the permissions of temporary files created by validate_cmd were different to the permissions defined on the file resource. PUP-8983

**Uninstall an rpm package without specifying a version or build number**

You can now remove a package with `rpm` package provider using `ensure => absent` without specifying a version or build number. PUP-8664

© 2022 Puppet, Inc., a Perforce company
Tags specified via --skip_tags are no longer expanded by splitting on the namespace

Using --skip_tags split tags on the namespace separator (::) and caused Puppet to expand and skip all resources in the specified namespace. For example, using fruit:apples expanded and skipped ['fruit::apples', 'fruit', 'apples']. PUP-8215

Improved error message when listing provider resources

You now get a more helpful error message if you try to list resources for a provider type that does not have the instances class method defined. PUP-4930

Performing two or more rapid-fire Puppet runs no longer results in a race condition

When an additional Puppet run was triggered immediately after Puppet had requested a certificate from the primary server, the primary server receives the original Certificate Signing Request (CSR) and continuously return the original certificate, which wouldn't match the new keys generated by the second Puppet run. PUP-2958

Removing a user resource on Solaris 11 installations with home directory configurations

Previously, trying to remove a user resource on a Solaris 11 installation using a home directory configuration resulted in an error. PUP-9706

Hiera 3 lookups with convert_to keys

If you used a Hiera 3 lookup or Hiera handled an alias and the key was configured with convert_to, you'd get an error: "undefined method 'call_function' for Hiera::Scope". PUP-9693

Perform string to integer conversions on decimal strings with leading zeros

Converting a decimal string with leading zeros – for example, Integer("08", 10) – to an integer would result in an error. PUP-9689

puppet device always initializes SSL directories with the correct permissions

When initializing new device certificates, puppet device would sometimes set permissions in a way that prevented the pe-puppet user from reading some directories. PUP-9642.

The Windows package resource removes trailing whitespace

This release updates the Windows registry read method to replace null byte sequences with a space. This issue caused PuppetDB to discard updated facts from affected nodes. PUP-9639

Puppet no longer upgrades Debian upgrade packages before setting them on hold

Prior to this release, if you set a Debian package on hold with ensure => held and the package had a pending upgrade, Puppet installed the upgrade before locking the package. PUP-9564

Disabled Ruby 2.5.1 automatic HTTP retry mechanism

This Ruby mechanism could cause the same report to be submitted multiple times, increasing the load on the Puppet Server report processor. PUP-3905

System updates to Ruby in the Puppet agent conflicted with other software

This release fixes an issue where the gem update --system command used in the Puppet agent caused conflicts with software that depends on gems in Puppet's vendored Ruby directory, such as r10k. Now gem paths always contain the path for this directory, even after updating. PA-2628

© 2022 Puppet, Inc., a Perforce company
Dependency issues when installing tools that require gems

This release fixes an issue where incorrectly named spec files caused gem dependency lookup failures. If you tried to install tools that rely on gems such as Facter, Puppet and Hiera gem dependencies could not be referenced. PA-2670

Puppet 6.4.0
Released 26 March 2019

New features

HTTP certification requests

When run with debug, Puppet now prints the HTTP request and the response information. For example:

```
returned 200 OK
```

PUP-8769

Debug logging for the exec resource

This version introduces the following improvements to debug logging for the exec resource:

- Running the exec resource with --debug and --noop now prints a debug message with the command if checks prevent it from being executed. If command, onlyif, or unless are marked as sensitive, all commands are redacted from the log output. PUP-9357
- Puppet now gives a debug message when checking the existence of a file specified by creates. PUP-9511

New method: Puppet::FileSystem.replace_file

Use Puppet::FileSystem.replace_file to replace a file. If a mode is specified, it will always be applied to the file. Otherwise, if the file being replaced exists, its mode will be preserved. If the file doesn't exist, then the mode will default to 0640. This method supersedes Puppet::Util.replace_file, which will be deprecated in a future release. PUP-9499

SSL Improvements

This version introduces several features to improve Puppet agent's SSL subsystem, including the introduction of an SSL state machine. PUP-9459

The following SSL improvements have been made:

- Puppet no longer uses Puppet::SSL::Host. Puppet::SSL::Host will be deprecated in a future release. PUP-9459
- Puppet no longer saves its public key to disk, because the public key is derivable from its private key and is contained in its certificate. If you need to, you can extract the public key using `openssl rsa -in $(puppet config print hostprivkey) -pubout`. PUP-9459
- The puppet ssl, puppet device, and puppet agent applications are now the only applications that can initialize SSL. Puppet applications other than puppet agent, puppet device, and puppet ssl now raise an error if they attempt to make an SSL connection while the SSL bootstrap process is incomplete. PUP-9461 PUP-9459
- Added an API for loading certificates, keys, and certificate revocation lists (CRLs). PUP-9455
- Added an API for creating an SSLContext containing certificates and keys needed to make an SSL connection. PUP-9456
- Added a method to Puppet::Network::HttpPool to create an HTTPS connection using a specified SSLContext. PUP-9457
- Instead of using Puppet::SSL::HOST, puppet ssl now uses an SSL state machine to download certificate authority (CA) and certificate revocation list (CRL) bundles. PUP-9458
- Puppet preserves existing user and group behavior when saving SSL-related files. PUP-9463

© 2022 Puppet, Inc., a Perforce company
• The new puppet ssl bootstrap action submits a CSR and downloads the client certificate without running puppet agent -t. PUP-9556

SUSE Linux Enterprise Server support removed
This release of the puppet-agent package removes support for SUSE Linux Enterprise Server 11/12 s390x. PA-2489

Resolved issues

Ruby security patch in puppet-agent package
This puppet-agent package release includes a security patch for Ruby 2.5.3. To learn more about the CVEs that this patch address, see the Ruby security advisories. PA-2512

Resolved SSL issues
Improvements in the SSL subsystem (PUP-9459) have resolved the following issues:
• Puppet no longer conditionally sends its certificate signing request (CSRs) based on the presence or absence of the file on disk. Instead it generates and sends the CSR whenever it needs to check for a certificate. Puppet still saves the CSR to disk, but it never reads it back in. PUP-4568
• Puppet no longer downloads the CSR from the server, so it can never get into a state where it saves the wrong CSR to disk, causing it to be stuck. As a result, it's now possible to enable allow_duplicate_certs=true and have the agent submit a CSR with the same name as a previous instance of the node. The admin still needs to revoke the old cert and sign the new CSR in order for the agent to get its certificate. PUP-2354
• Puppet no longer uses the indirector to handle certificates or keys. PUP-6207
• Puppet was too permissive about skipping SSL verification if no client certificate was found. Puppet now never downgrades verification based on the absence of a client certificate. PUP-7295
• Mismatched certificates were cached on the host, causing Puppet to print an error on each run until an admin removed the files. If a client certificate, Certificate Authority (CA) bundle, or certificate revocation list (CRL) bundle are invalid, Puppet now discards them. PUP-7903
• The error message for a mismatched certificates name was not helpful. When a Puppet agent tries to connect to an SSL server where the certificate does not match the hostname it is trying to connect to, it will now return the expected error message. PUP-8213
• The Puppet agent was not verifying its peer in an SSL connection when downloading a CRL. Puppet now verifies the server's SSL certificate when retrieving a CRL. PUP-9142

filebucket type server and port settings no longer have explicit defaults
For the filebucket type, server and port no longer have explicit default values in the type definition. If server is not set, it defaults to the first entry in server_list if set; otherwise, it defaults to server. If port is not set, it defaults to the port in the first entry of server_list if set; otherwise, it defaults to serverport. PUP-9025

Custom functions can now be correctly called
This release fixes an issue where the call () function could call only functions that existed in Puppet core; custom functions could not be called. Now any function in the environment is visible and can be called. PUP-9477

Puppet agent now produces an error when a functional server is not found
If server_list is set and a functional server is not found, Puppet returns an error rather than falling back to the server setting. PUP-9076
Optional type without arguments no longer returns an error

Previously, if you used the type `Optional` without any arguments, it could result in an internal error. This is now fixed. On its own, `Optional` means the same as `Any`. Supply a type argument with the desired type if the value is not `undef`. PUP-9467

Fixed remote MSI package installation on Windows

This release fixes a regression that prevented installing MSI packages from an HTTP URL on Windows. PUP-9496

Puppet 6.3.0
Released 20 February 2019

New features

Corrective changes explicitly logged

Prior to this release, agent runs provided the same output for both intentional and corrective changes. Now corrective changes are now explicitly called out in the logs as corrective. PUP-9324

New collections and index functions

This release adds `group_by` and `partition` functions that operate on collections.

The `group_by` function is useful for separating targets into categories based on properties of those targets. The `partition` function allows you to use a Boolean condition to separate targets that otherwise be two opposite filter operations. PUP-9372

This release also adds an `index()` function that can return the index of a specified element in an Array, Hash, String, or other iterable value that is either equal to a given value or for which a given lambda returns a truthy value. PUP-9482

Puppet-agent package added for macOS 10.14 Mojave

This release adds a `puppet-agent` package for macOS 10.14 Mojave.

On macOS 10.14 Mojave, you must grant Puppet Full Disk Access to be able to manage users and groups. To give Puppet access on a machine running macOS 10.14, go to System Preferences > Security & Privacy > Privacy > Full Disk Access, and add the path to the Puppet executable. Alternatively, set up automatic access using Privacy Preferences Control Profiles and a Mobile Device Management Server. PA-2226, PA-2227

Resolved issues

Syntax errors on interpolated heredocs are resolved

Heredoc expressions with interpolation using an access expression such as `$facts['somefact']` sometimes failed with a syntax error. This error was related to the relative location of the heredoc and surrounding whitespace and is now resolved. PUP-9303

Restarting `pxp-agent` service kills all processes when restarted

This release modifies the `pxp-agent` service to kill all `pxp-agent` processes when the service is restarted, rather than only the current process. PCP-833

Custom functions can now be correctly called

This release fixes an issue where the `call()` function could call only functions that existed in Puppet core; custom functions could not be called. Now any function in the environment is visible and can be called. PUP-9477
Optional type without arguments no longer returns an error

Previously, if you used the type `Optional` without any arguments, it could result in an internal error. This is now fixed. On its own, `Optional` means the same as `Any`. Always supply a type argument with the desired type if the value is not `undef`. PUP-9467

Invalid path to `--logdest` option was ignored

Now if you give a `--logdest` location that Puppet cannot find or write to, the run fails with an error. PUP-6571

Upstart provider evaluated during provider check instead of during loading

The upstart provider was being evaluated when loaded, causing issues with testing and availability during transactions. This has been fixed so that the provider is evaluated only when provider suitability is being checked. PUP-9336

Deprecations

Fine grained control of file and environment timeouts deprecated

Fine grained control of file and environment timeouts is deprecated. Instead, use `0` or `unlimited` to control default caching behavior and the environment-cache endpoint in Puppet Server's administrative API to expire the cache as needed. PUP-9497

`SublocatedExpression` class

The AST `SublocatedExpression` class is no longer generated by the parser. The `SublocatedExpression` class itself will be removed from Puppet in a future release. PUP-9303

Puppet 6.2.0

Released 24 January 2019

New features

Improved support for Solaris `pkg` provider

This release adds support for `install_options` for the `pkg` provider on Solaris. PUP-5287

`puppet-agent` package available for Red Hat 8 beta

A `puppet-agent` package is available for the beta release of Red Hat 8. Because this Red Hat version is a beta release, not all functions are verified to work correctly. Do not use this version of `puppet-agent` in production.

Added protection against illegal methods in legacy functions

Puppet now protects against illegal method definitions in loaded legacy functions. Illegal methods in legacy functions disrupt the entire system and can cause difficult-to-diagnose issues. For information on how to remove such methods from legacy functions, see Refactoring legacy 3.x functions. PUP-9294

Puppet logs JSON in newline-delimited format

For newline-delimited JSON logs, set the `logdest` setting or command line option to an absolute path ending with the `.jsonl` extension, such as `puppet agent --logdest /var/log/puppet.jsonl`. This is preferable to the JSON format, since the generated log is not valid JSON and requires an appended closing `]` to make it valid. PUP-8687

Fedora 27 `puppet-agent` package no longer available

Fedora 27 reached end of life in November 2018. Updated `puppet-agent` packages for this version are no longer available.
Resolved issues

Fix for specifying package versions in portage provider
Prior to this release, you could not specify package versions when managing packages with the portage package provider. The provider now accepts and honors a version string for managing packages. PUP-9071

Exceptions encountered during resource pre-fetch are logged
If the Puppet agent encounters exceptions when pre-fetching resources for catalog application, it now logs the exceptions and returns a more useful error message. PUP-8962

Puppet commands fail if the puppet.conf file is unreadable
Puppet commands now fail if Puppet Server is unable to read the puppet.conf file. Only the --help and --version commands work if the puppet.conf file is unreadable. PUP-5575

Improved error handling for PNTransformer
When parsing Puppet into structured AST, the Puppet parser produced an error on some empty constructs because the PNTransformer could not resolve them. Now it generates a Nop expression instead. PUP-9400

Failed dependency resources are reported only once
After a failed resource has been reported, other resources that depend on the failed resource will not be reported again. However, you still get the skip message for each skipped resource. PUP-6562

Command line module installation improved
The puppet module install command now downloads only the release metadata it needs to perform dependency resolution, drastically reducing data download and improving installation time. For the puppetlabs-stdlib module, this change reduces the data download from 25MB to 68KB, and any module that depends on stdlib installs faster. PUP-9364

Puppet no longer ignores the srv_domain setting
This release fixes an issue where Puppet 6.0 ignored the srv_domain setting when using DNS SRV records to connect to the ca_server. PUP-9399

Puppet 6.1.0
Released 18 December 2018

New features

Package provider included in puppet resource default output
The puppet resource <package-name> command now includes the package provider as part of its output.

Clean certificates for remote nodes
The puppet ssl clean command now accepts a --target CERTNAME parameter to clean certificates for remote nodes configured through device.conf. PUP-9248

puppet-agent package available for Fedora 29
A puppet-agent package is now available for Fedora 29.
The systemd provider checks for new and changed units

When Puppet starts or restarts a service, the systemd provider checks to see whether the service requires a daemon reload, and reloads the systemd daemon if necessary. This ensures the service runs with the latest configuration on disk. PUP-3483

New service timeout parameter for Windows

On Windows systems, you can now specify a service timeout parameter that configures how long, in seconds, Puppet should wait when syncing service properties. PUP-9132

For example:

```
service { "service-name":
  enable => true,
  ensure => running,
  timeout => 45
}
```

Format strings with double quotes

You can now force strings to have double quotes when formatting strings with the `String.new` function. Previously, this function dynamically determined whether to use single quotes or double quotes for the string.

To force double quotes, specify the alternative format flag `#` when you specify the string format. PUP-9344

Service support for systemd on Linux Mint 18 and newer

This release adds support for services on Linux Mint 18 and newer, which use the init systemd instead of upstart for services. PUP-9326

Parser validation finds and reports syntax issues in heredocs

Heredoc syntax checking is now performed during validation, unless the heredoc contains any interpolated expressions. This includes manifest and EPP template syntax checks. Heredocs with interpolation are checked during evaluation. Prior to this release, heredoc syntax checks were run only during evaluation.

By default, heredocs are treated as text unless otherwise specified with the end-of-text tag, such as `@(EOT:json)`, `@(EOT:pp)`, or `@(EOT:epp)`. To check syntax in heredocs, run `puppet parser validate`. PUP-9257

Resolved issues

Improved handling of incompletely configured services

Puppet now treats incomplete services the same way as nonexistent services, returning `enabled => false` and `ensure => :stopped` in either case. If you try to set `ensure => running` or `enabled => true` on an incomplete or nonexistent service, Puppet raises an error. PUP-9240

Refreshed resource status now included in event report

This release fixes an issue where refreshed resources, such as reboot or some execs, did not create a status event in the event report. PUP-9339

Catalog compilation error fixed

When compiling a catalog, Puppet sometimes raised the error "Attempt to redefine entity." This issue has been fixed with an update to the internal logic. PUP-8002
The exec provider supports empty environment variables

Prior to this release, the exec provider did not support empty environment variables. If an empty environment variable
is set on the system, Puppet temporarily overwrites it during execution. PUP-1209

Puppet recognizes tmpf$s support for SELinux labels

SELinux utilities within the Puppet codebase now recognize that the tmpf$s supports extended attributes and
SELinux labels. The query selinux_label_support? returns true for a file mounted on tmpf$s. PUP-9330

Fixed recognition of short form Arrays and Hashes

This release fixes a regression in the string formatting rules that caused a short form for an Array or Hash to not
be recognized. For example, String([1, [2, 3], 4], '%#a") would not format with indentation, but would
format the long form String([1, [2, 3], 4], (Array => ( format => '%#a"))). Now the short form
works for Array and Hash as intended. PUP-9329

puppet ssl clean now deletes local certificate requests

This release fixes an issue where the puppet ssl clean command did not correctly delete local certificate
requests. PUP-9327

Resolved Timestamp and Timespan data type errors

Prior to this release, the data types Timestamp and Timespan raised errors if time range was specified with
Integer or Float values. These data types now support time ranges specified with these values. PUP-9310

Puppet 6.0.0
Released 18 September 2018

New features

Select types moved to modules

In this release, many types were moved out of the Puppet codebase (stdlib), and into modules on the Forge. This
change enables easier composability and reusability of the Puppet codebase and enables development to proceed more
quickly without risk of destabilizing the rest of Puppet. Some types are now in supported modules and are repackaged
back into the agent. Some are now in modules that are updated, but are not repackaged into the agent. And some are
in modules that are deprecated, not updated, and not repackaged back into the agent.

Note: New functions handle undef values more strictly than their stdlib counterparts. Code that relies on undef
values being implicitly treated as other types will return an evaluation error.

See the Resource Type Reference for the full list and links to the Forge modules for those types that moved.

Resource API

Resource API has been added, providing a new, recommended method to create custom types and providers. The
Resource API is built on top of the Puppet core and is easier, faster, and safer than the old types and providers
method. See the Resource API documentation for more information.

Upgrade to Ruby 2.3 or later

Puppet now requires Ruby 2.3 or later, and reports an error when running older Ruby versions. We removed code
paths for older Ruby support, such as 1.8.7, relaxed our gem dependencies to include gems that require Ruby 2 or up,
and now test Puppet pull requests against JRuby 9k. PUP-6893, PUP-8483, PUP-8484

© 2022 Puppet, Inc., a Perforce company
**puppetserver ca command**

The new `puppetserver ca` command replaces the `puppet cert` command. Running the `puppet cert` command results in an error, with instructions on alternative commands to use. Some actions (`fingerprint`, `print`) have not been directly replaced, because OpenSSL provides good equivalents. For verifying certificates, use `puppet ssl verify`. For more usage details, see the intermediate CA documentation. PUP-9022

**node clean uses the Puppet Server CA API**

The `puppet node clean` command now goes through the Puppet Server CA API to clean up certificates for a given node. This avoids issues where multiple entities attempt to revoke certs at once, because all of these updates are now funneled through the API, which handles concurrent requests correctly. See SERVER-115 and PUP-9108 for more information.

**Agents can use CA and CRL bundles**

The agent now correctly saves and loads chained SSL certificates and certificate revocation lists when in an environment where its certificates are issued by Puppet acting as an intermediate CA. PUP-8652

**Load files from pluginsync during catalog application**

Use the `Deferred` data type in a catalog to call functions on the agent before the catalog is applied. It is now possible to call all functions implemented in Ruby on the agent side. It is not possible to call functions written in the Puppet language, as they are not available on the agent. PUP-9035

**puppet ssl subcommand**

There is a new Puppet subcommand for working with SSL certificates. The `puppet ssl` command supports the `submit_request`, `download_cert`, and `verify` actions for working with SSL certificates on the agent. PUP-9028

**File requirements included in task infoservice files responses**

When requesting task details, the primary server now returns a list of all files from the tasks metadata `files` and `implementations['files']` keys. PUP-9081

**Devuan default service provider**

The Devuan service provider now defaults to the Debian init provider. PUP-9048

**apply statement in plan language**

An `apply` keyword has been added to the Puppet parser when running with tasks enabled. See Puppet specifications for details. PUP-8977

**Updated default input_method for task object type**

The `input_method` property of tasks now defaults to `undef` rather than the string `both`. This allows more flexibility in defaults and what `input_methods` we choose to support in the future. PUP-8898

**convert_to() function accepts additional arguments**

`convert_to()` function now accepts additional arguments. Previously, it accepted only the data type to convert to. PUP-8761
**compare function**

A compare\((a, b)\) function has been added. It returns \(-1, 0,\) or \(1\) depending on if \(a\) is before \(b\), same as \(b\), or after \(b\). The function works with the comparable types: \texttt{String}, \texttt{Numeric}, \texttt{Semver}, \texttt{Timestamp}, and \texttt{Timespan}. For \texttt{String} comparison it is possible to ignore or take case into account. PUP-8693

**Deferred data type**

A new data type \texttt{Deferred} has been added. It is used to describe a function call that can be invoked at a later point in time. PUP-8635

**Call function resolves Deferred values**

It is now possible to resolve a \texttt{Deferred} value by using the \texttt{call} function. It can resolve a \texttt{deferred} function call, and a \texttt{deferred variable dereference (with support to dig into a structured value)}. PUP-8641

**Concatenate with + on Binary type**

It is now possible to use the plus operator + to concatenate two \texttt{Binary} data type values. (PUP-8605) The \texttt{sort()} function has been moved from \texttt{stdlib} to Puppet. The function now also accepts a lambda for the purpose of using a custom compare. PUP-8622

**Select string functions moved from stdlib to Puppet**

The functions \texttt{upcase()}, \texttt{downcase()}, \texttt{capitalize()}, \texttt{camelcase()}, \texttt{lstrip()}, \texttt{rstrip()}, \texttt{strip()}, \texttt{chop()}, \texttt{chomp()}, and \texttt{size()} have been updated to the modern function API and the new versions are in Puppet and no longer require \texttt{stdlib}. The functions are generally backward compatible. PUP-8604

**Select math functions moved from stdlib to Puppet**

The math functions \texttt{abs}, \texttt{ceil}, \texttt{floor}, \texttt{round}, \texttt{min}, and \texttt{max} are now available in Puppet. The functions are compatible with the functions with the same name in \texttt{stdlib} with the added feature in \texttt{min} and \texttt{max} of calling them with a single array and being able to use a lambda with a custom compare. These \texttt{stdlib} math functions used inconsistent string to numeric conversions that were also unintentionally making the functions compare values in strange ways. The automatic conversions are now deprecated and will issue a warning. PUP-8603

**Agent support for rich data content negotiation**

The \texttt{rich_data} setting is now enabled by default. Catalog requests have two new content types, \texttt{application/vnd.puppet.rich+json} and \texttt{application/vnd.puppet.rich+msgpack}, that are used when both primary server and agent have this enabled (and depending on whether \texttt{preferred_serialization_format} is \texttt{json} or \texttt{msgpack}). PUP-8601

**vendor_modules added to basemodulepath**

The default \texttt{basemodulepath} now includes a vendored modules directory, which enables Puppet to load modules that are vendored in the puppet-agent package. To prevent Puppet from loading modules from this directory, change the \texttt{basemodulepath} back to its previous value, for example, on *nix: \$\texttt{codedir/modules:/opt/puppetlabs/puppet/modules}. On Windows: \$\texttt{codedir/modules}. PUP-8582

**environment.conf modulepath accepts globs**

The \texttt{modulepath} as defined in \texttt{environment.conf} can now accept globs in the path name. PUP-8556

**Customize default package providers**

This change adds a \texttt{notdefaultfor} that prevents a provider from being a default for a given set of facts. \texttt{notdefaultfor} overrides any \texttt{defaultfor} and should be defined more narrowly. PUP-8552

© 2022 Puppet, Inc., a Perforce company
Define properties or parameters for types as sensitive
Parameters can now be marked sensitive at the class level rather than just the instance level. PUP-8514

Update default provider for Ubuntu
SystemD is now the default provider for Ubuntu 17.04 and 17.10. PUP-8495

Functions to use dot notation to dig into a hash or array
It is now possible to use dot notation to dig out a value from a structure, like in Hiera lookup and elsewhere in Puppet. To support this, the `getvar()` function has moved from `stdlib` to Puppet, and we have added a new function `get()`. You can now for example use `getvar('facts.os.family')` starting with the variable name. The `get` function is the general function which takes a value and a dot-notation string. PUP-7822

Puppet 4.x functions available to all modules
It is no longer required to have a dependency listed in a module’s `metadata.json` on another module in order to use functions or data types from the other module. PUP-6964

Updated addressable Ruby gem
Updated the version of the addressable Ruby gem now that JRuby 1.9.3 support has been removed. PUP-6984

Undeprecated certificate authority settings
Settings related to certificate authorities are no longer being deprecated as planned in PUP-9027. Warnings related to these planned deprecations have been removed. PUP-9116

Resolved issues

Reported events didn’t stringify rich data
With rich data turned on for a catalog (now the default), a report could contain rich data in reported events, but nothing downstream from the agent was prepared to handle rich data. This is now fixed so that data in reported events are stringified when needed. PUP-9093

Illegal top-level constructs didn’t produce an appropriate error
The deprecation for illegal top-level constructs is now an error. PUP-9091

__ptype and __pvalue were allowed as attribute names
Attempt to use the reserved attribute names __ptype and __pvalue in custom Object data types now raises an error instead of producing a bad result when serializing such objects. PUP-9079

A hash containing the key __ptype couldn’t be serialized using human-readable JSON
It was not possible to use a hash key __pcore_type in a hash because that triggered the special handling during serialization. Now, the special key has changed to __ptype and it is not possible to also use that as a key in a hash and still be able to serialize it, such as using it in a catalog. PUP-8976

Status endpoint wasn’t used to determine if Puppet Server was available
When the agent is configured with a list of servers (using `server_list`), it now requests server status from the status endpoint instead of the node endpoint. PUP-8967

© 2022 Puppet, Inc., a Perforce company
Selmodule thought ‘foo’ existed if ‘myfoo’ was loaded

The selmodule type is more strict about checking if a module has already been loaded, and no longer considers modules such as "bar" and "foobar" to be the same module. PUP-8943

Resource status of failed_to_restart wasn’t included in reports

Puppet considers resources that have failed to restart when notified from another resource as failed, and marks them as such in reports. Reports also now include the failed_to_restart status for individual resources, instead of only including a total count of failed_to_restart resources in the resource metrics section. This bumps the report format version to 10. PUP-8908

File type wasn’t redefined when required in code

Fixed an issue running in JRuby where we didn’t store autoloaded paths in the same way that the JRuby implementation did, leading to a bug where a type or provider could get loaded more than once. PUP-8733

Puppet lost track of the current environment

Puppet autoloader methods now require a non-nil environment. This is a breaking API change, but not affect any user extensions like 3x functions. Puppet sometimes used the configured environment instead of the current environment to autoload. This mainly affected agents when loading provider features. Calling Puppet::Parser::Functions.autoloader.load* methods are deprecated, and issue a warning if strict mode is set to warning or error. Instead use Scope#call_function("myfunction") to call other functions from within a function. PUP-8696

Comparison of numeric to timestamp or timespan failed

When comparing numeric to timestamp or timespan it did not work to compare with the numeric value first. This is now fixed. PUP-8694

http_read_timeout and runtimeout defaults were limited

The http_read_timeout default changed from infinite to 10 minutes. This prevents the agent from hanging if there are network disruptions after the agent has sent an HTTP request and is waiting for a response that might never arrive. Similarly, the runtimeout default also changed from infinite to 1 hour. PUP-8683

tidy resource was too chatty

The tidy resource type now uses the debug log level for its File does not exist message, instead of the info level. This means that resources of this type will no longer emit the message by default when the target of the resource has already been cleaned from disk. PUP-8667

Agents didn’t use the CRL bundle to verify primary server revocation status

With this change, if the user has distributed the CRL chain out-of-band, then the agent successfully loads it and uses it to verify its connection to other Puppet infrastructure (for example, the primary server). It expects the CRL chain to be one or more PEM-encoded CRLs concatenated together (the same format as a cert bundle). This fixes the "Agent-side CRL checking is not possible" caveat in Puppet 5. PUP-8656

Puppet 5 and Ruby 2.4 couldn’t handle invalid plists

When processing malformed plist files, we used to use /dev/stdout, which can cause Ruby to complain. We now use – instead which means to use stdout when processing the plist file with plutil. PUP-8545

© 2022 Puppet, Inc., a Perforce company
White space surrounding comments in EPP were handled incorrectly

EPP comments `<%# Like this %>` always trimmed preceding whitespace. This is different from ERB making it more difficult to migrate ERB templates to EPP. There was also no way of making EPP preserve those spaces. Now, EPP comment does not trim preceding whitespace by default, and a new left trimming tag `<%#` has been added. This is a backwards incompatibility in that code such as "Before `<%# comment %>` after" resulted in the string "Beforeafter", whereas now it will be "Before after". PUP-8476

filter function didn’t accept truthy values

The filter function did not accept truthy value returned from the block as indication of values to include in the result. Only exactly Boolean `true` was accepted. PUP-8320

Unsafe YAML data could be loaded

Puppet now uses `YAML.safe_load` consistently to ensure only known classes are loaded. PUP-7834

Gemfile didn’t support consuming the Puppet gem via a Git reference

Restructure the Puppet Gemfile so that bundler installs Puppet’s runtime, feature-related, and test dependencies by default. The development and documentation groups can be installed using: `bundle install --with development documentation`. PUP-7433

Puppet didn’t use shared gem dependency for semantic_puppet

Puppet now uses the shared gem dependency for `semantic_puppet`, rather than loading its own vendored version. PUP-7157

Puppet, the agent, and Puppet Server didn’t use a shared gem directory for semantic_puppet

Puppet now loads `semantic_puppet` from a shared gem directory, so that Puppet, the agent, and Puppet Server all require and use the same version of the gem. ( ~>1.0.x) PUP-7115

metrics.time.total didn’t correlate to time reported in log

Total time now reports the measured time of the run instead of a sum of other run times. PUP-6344

Features weren’t re-evaluated when a block was used

Features defined using a block or a list of libraries now behave the same, so the following are equivalent:

```ruby
Puppet.features.add(:my_feature) do require 'mylib' end
```

and

```ruby
Puppet.features.add(:my_feature, libs: ['my_lib'])
```

Previously the result of the block was always cached. With this change only true or false return values are cached. To indicate that the state of the feature is unknown and may become available later, the block returns nil. PUP-5985

Manifests that declared things in the wrong namespace didn’t trigger errors

Errors will be reported for module files declarations that have a namespace inconsistent with their directory and file location. PUP-4242

Invalid .dot files were generated by missing escapes of quoted strings

Generating graphs of catalogs (such as `puppet apply --graph`) now correctly handles resources with double quotes in the title. PUP-2838
Outdated vendored semantic_puppet gem

We have updated the vendored semantic_puppet gem in the puppet-agent package to the most recent version, 1.0.2. PA-1881

Deprecations

Certificate authority subcommands and v1 CA HTTP API

Certificate authority subcommands have been removed from Puppet, including: cert.ca, certificate, certificate request, and certificate_revocation_list. Use puppetserver ca and puppet ssl instead. PUP-8998

As a part of the larger CA rework, the v1 CA HTTP API is removed (everything under the ca url /v1). PUP-3650

Ruby certificate authority

Puppet no longer has a Ruby CA. All CA actions now rely entirely on the Clojure implementation in Puppet Server. It can be interacted with by means of the CA API and the puppetserver ca command, which leverages the API using subcommands like those provided by puppet cert. PUP-8912

Trusted server facts

Trusted server facts are always enabled and the trusted_server_facts setting has been deprecated since 5.0. This removes the setting and conditional logic. PUP-8530

write_only_yaml node terminus

The write_only_yaml node terminus was used to “determine the list of nodes that the primary server knows about” and predated widespread PuppetDB adoption. The write_only_yaml has been deprecated since 4.10.5, and this commit removes it. Note this results in a Puppet Server speedup as it no longer needs to serialize node data as YAML to disk during a compile. PUP-8528

LDAP node terminus

The LDAP node terminus has been removed. PUP-7601

computer, macauthorization, and mcx types and providers

The computer, macauthorization, and mcx types and providers have been moved to the macdslocal_core module. It is not repackaged into puppet-agent in the 6.0 series.

Nagios types

The Nagios types no longer ship with Puppet, and are now available as the puppetlabs/nagios_core module from the Forge.

Cisco network devices

The Cisco network device types no longer ship with Puppet. These types and providers have been deprecated in favor of the puppetlabs/cisco_ios module, which is available on the Forge. PUP-8575

:undef in types and providers

In previous versions, values from manifests assigned to resource attributes that contained undef values nested in arrays and hashes would use the Ruby symbol :undef to represent those values. When using puppet apply types and providers would see those as :undef or as the string “undef” depending on the implementation of the type. When using a primary server, the same values were correctly handled. In this version, Ruby nil is used consistently for this. (Top level undef values are still encoded as empty string for backwards compatibility). PUP-9112
**puppet module build command**

To reduce the amount of developer tooling installed on all agents, this version of puppet removes the `puppet module build` command. To continue building module packages for the Forge and other repositories, install Puppet Development Kit (PDK). PUP-8763

**pcore_type and pcore_value**

The earlier experimental `-rich_data` format used the tags `pcore_type` and `pcore_value`, these are now shortened to `__ptype` and `__pvalue` respectively. If you are using this experimental feature and have stored serializations you need to change them or write them again with the updated version. PUP-8597

**Webrick**

Webrick support (previously deprecated) has been removed. To run Puppet as a server you must use Puppet Server. PUP-8591)

**puppet server command**

The `puppet server` command and its subcommands have been removed. Instead, use a `puppet-config` command. PE-24280

**–strict flag in puppet module command**

The `–strict` flag in `puppet module` has been removed. The default behavior remains intact, but the tool no longer accepts non-strict versioning (such as release candidates and beta versions). PUP-8558

**Select settings**

The following settings have been removed:

- The previously deprecated `configtimeout` setting has been removed in favor of the `http_connect_timeout` and `http_read_timeout` setting. PUP-8534
- The unused `ignorecache` setting has been removed. PUP-8533
- The previously deprecated `pluginsync` setting has now been removed. The agent’s pluginsync behavior is controlled based on whether it is using a cached catalog or not. PUP-8532
- The deprecated `app_management` setting has now been removed. Previously, this setting was ignored, and always treated as though it was set to be on. PUP-8531
- The deprecated `ordering` setting has been removed, and catalogs now always have the ordering previously provided by the `manifest` value of this setting. PUP-6165
- Settings related to the rack webserver from Puppet, including `binaddress` and `serverhttplog`. PUP-3658

**String duplication in 3x runtime converter**

Types and provider implementations must not mutate the parameter values of a resource. With this release, it is more likely that the parameters of a resource have frozen (that is, immutable) string values and any type or provider that directly mutates a resource parameter might fail. Previously, every resource attribute was copied to not make application break even if they did mutate. Look for use of `gsub!` in your modules and replace logic with non-mutating version, or operate on a copy of the value. All authors of Forge modules having this problem have been notified. PUP-7141

**Puppet.newtype method**

The deprecated `Puppet.newtype` method (deprecated since 2011) has now been removed. (PUP-7078)
Certificate handling commands deprecated but not removed

The following subcommands were deprecated in a previous version and slated for removal in this version. While these subcommands are still deprecated, they have not yet been removed.

- ca_name
- cadir
- cacert
- cakey
- capub
- cacrl
- caprivatedir
- csrdir
- signeddir
- capass
- serial
- autosign
- allow_duplicate_certs
- ca_ttl
- cert_inventory

Puppet known issues

These are the known issues in this version of Puppet.

Puppet lookup omits parameters when using --environment

If you specify puppet lookup with an explicit environment using the --environment flag, puppet lookup does not call to the ENC, causing any node parameters set in the ENC to be omitted. PUP-11595

Puppet lookups fail to interpolate toplevel variables when an environment is specified

In Puppet 6.26 and 7.14, the lookup command fails to resolve toplevel facts in hiera configs if you're using the --environment option. For example, if you use a toplevel variable like "nodes/%{fqdn}.yaml", Puppet interpolates the variable as an empty string. As a workaround, use trusted facts or specify the fact value using the "facts" hash, such as "${facts.hostname}" PUP-11437

User and group management on macOS 10.14 requires Full Disk Access

To manage users and groups with Puppet on macOS 10.14, you must grant Puppet Full Disk Access (FDA). You must also grant FDA to the parent process that triggers your Puppet run. For example:

- To run Puppet in a server-agent infrastructure, you must grant FDA to the pxp-agent.
- To run Puppet from a remote machine with SSH commands, you must grant FDA to sshd.
- To run Puppet commands from the terminal, you must grant FDA to terminal.app.

To give Puppet access on a machine running macOS10.14, go to System Preferences > Security & Privacy > Privacy > Full Disk Access, and add the path to the Puppet executable, along with any other parent processes you use to run. For detailed steps, see Add full disk access for Puppet on macOS 10.14 and newer. Alternatively, set up automatic access using Privacy Preferences Control Profiles and a Mobile Device Management Server. PA-2226, PA-2227

Hiera knockout_prefix is ineffective in hierarchies more than three levels deep

When specifying a deep merge behaviour in Hiera, the knockout_prefix identifier is effective only against values in an adjacent array, and not in hierarchies more than three levels deep. HI-223

© 2022 Puppet, Inc., a Perforce company
Specify the epoch when using version ranges with the **yum** package provider

When using version ranges with the **yum** package provider, there is a limitation which requires you to specify the epoch as part of the version in the range, otherwise it will use the implicit epoch ‘0’. For more information, see the [RPM packaging guide](#). **PUP-10298**

Deferred functions can only use built-in Puppet types

Deferred functions can only use types that are built into Puppet (for example **String**). They cannot use types from modules like **stdlib** because Puppet does not plugin-sync these types to the agent. **PUP-8600**

The Puppet agent installer fails when systemd is not present on Debian 9

The **puppet-agent** package does not include sysv init scripts for Debian 9 (Stretch) and newer. If you have disabled or removed systemd, **puppet-agent** installation and Puppet agent runs can fail.

Upgrading Windows agent fails with `ScriptHalted` error

Registry references to `nssm.exe` were removed in **PA-3263**. Upgrading from a version without this update to a version that contains it triggers a Windows SecureRepair sequence that fails if any of the files delivered in the original `*.msi` package are missing. This is an issue when upgrading to one of the following Puppet agent versions: 5.5.21, 5.5.22, 6.17.0, 6.18.0, 6.19.0, 6.19.1, 6.20.0, 7.0.0, 7.1.0 or 7.3.0. To work around this issue, put the `*.msi` file for the currently installed version in the `C:\Windows\Installer` folder before you upgrade. Starting with Puppet agent 6.21.0 and 7.4.0, the `nssm.exe` registry value will be replaced with an empty string, instead of the registry key being removed, to avoid triggering Windows SecureRepair. **PA-3545**

Puppet Server release notes

**Puppet Server 6.20.0**

Released August 2022 and shipped with Puppet 6.28.0.

Enhancements

**puppetserver http client respects `include_system_store` option.** Puppet Server’s Ruby HTTP client now supports loading certificates from the system trust store that is included with Puppet Agent. It also supports loading certs from a file or Java cert store at an arbitrary location via the `ssl_trust_store` setting. **SERVER-2944**

Resolved issues

**RPM should create puppet user with UID/GID 52.** When the `puppet` user and group are created on rpm-based systems, they are now assigned a static UID/GID of 52. **SERVER-1381**

**Puppet Server 6.19.0**

Released April 2022 and shipped with Puppet 6.27.0.

Enhancements

**Enable sles-15-x86_64 builds and testing for puppetserver.** We now support **puppetserver** on sles-15-x86_64. **SERVER-3156**

**Bump BouncyCastle to 1.70.** Puppet Server now ships with Bouncy Castle 1.70, which has improved TLS 1.3 support. **SERVER-3135**

**JRuby pool lock lifecycle logging.** The JRuby lock lifecycle of request, acquire, and release is now logged at the INFO level, rather than DEBUG. **SERVER-3098**

Resolved issues

**Bad exit code for errors in `puppetserver ca list`**. The `puppetserver ca list` command will now exit 1 when run on a non-CA server. **SERVER-2797**
**Puppet Server CA always creates type 1 authority key identifiers.** Previously, Puppet Server would always compute a type 1 key identifier based on the public key of the certificate authority. This is incompatible in situations where Puppet Server imports pre-made certificates that use a type 2 key identifier.

Now, Puppet Server will copy the subject key identifier from the ca certificate instead of computing a type 1 key identifier. This will allow for type 2 identifiers and future key types on the CA. With this change, Puppet Server can now use an intermediate certificate authority signed by HashiCorp’s Vault or AWS ACM. SERVER-2662

**Puppet Server 6.18.0**

Released January 2022 and shipped with Puppet 6.26.0.

**New features**

- **Metrics collection with Dropsonde.** Users can now enable module metrics collection via Dropsonde. To turn this on, configure `dropsonde: { enabled: true }` in `puppetserver.conf`. By default when enabled, Dropsonde collects metrics when the service is started and once a week thereafter. SERVER-3079

**Resolved issues**

- **CRL uploading.** The CRL update endpoint will now issue a meaningful error message when a CRL without an authority key identifier is sent in the request body. SERVER-3080
- **CA Authority Key Identifier incorrectly filled with issuer instead of keyid.** The self-signed CA signing cert generated by starting puppetserver will now use a `keyid` for its authority key identifier to match the CA chain generated by `puppetserver ca setup`. SERVER-3114
- **CA added a Subject Alternative Name extension to CA certs** The CA signing cert no longer has subject alternative names added to it, since they are not meaningful. SERVER-3114

**Puppet Server 6.17.1**

Released November 2021 and shipped with Puppet 6.25.1.

This release includes security fixes. For the latest features, see the release notes for Server 6.17.0.

**Puppet Server 6.17.0**

Released October 2021 and shipped with Puppet 6.25.0.

**Enhancements**

- **Retrieve facts from any terminus.** The v4 catalog endpoint (used by Impact Analysis) now supports retrieving facts from any facts terminus, if none are provided with the request.
- **TLS 1.3 support.** Puppet Server now supports TLS 1.3 and associated cipher suites by default. SERVER-3076
- **Improved performance in list command.** Puppetserver CA CLI command `list` utilizes the updated `certificate_status` endpoint for a faster performance when listing certificate requests. SERVER-3060
- **--force flag in certificate generation.** The `puppetserver ca generate --ca-client` command can now take a `--force` flag, which forces the tool to generate the certificate even if it cannot determine whether Puppet Server is offline. To avoid CA corruption, ensure your server is offline before you use this flag. SERVER-2842
- **Prune duplicate entries from CRL.** In this release, the `puppetserver ca` subcommand now accepts the `prune` actions. These actions allow you to prune any duplicate certificates from Puppet’s CRL. SERVER-2740
- **CRL query speed.** Querying CRLs from `puppetserver` is now faster, resulting in fewer timeouts. SERVER-3020
- **Scripts as a default mount.** Puppet Server has a new default mount named `scripts`. You can use the new default mount with API endpoints such as `file_content`, `file_metadata`, and `static_file_content` to load scripts from the `scripts/` directory of a module. SERVER-3058

**Resolved issues**

- **Duplicate entries in CRL.** Puppet CA no longer allows adding duplicate certificates to the CRL. SERVER-2509
Puppet Server 6.16.1

Released July 2021 and shipped with Puppet 6.24.0.

Enhancements

• **Add `--verbose` flag.** In this release, the `puppetserver ca` subcommand now accepts the `--verbose` flag. If the `--verbose` flag is passed, it displays additional low-level details about the invoked action (such as details about HTTP requests created by the tool). SERVER-2251

• **Specify certificate output in JSON.** In this release, the `puppetserver ca list` action now accepts a `--format` flag that can be used to display certificates in JSON format. The output format is text by default. SERVER-3006

• **Jetty 9.4.42.** This release includes a Jetty update to 9.4.42. SERVER-3035

Resolved issues

• **CRL update endpoint is not enabled by default.** The PUT /puppet-ca/v1/certificate_revocation_list endpoint is now enabled by default for clients that have a special cert extension. Previously, you had to manually update the auth.conf file to access this endpoint. SERVER-3033

• **Puppet Server cannot use OpenSSL EC files in OpenSSL format.** Previously, Puppet Server failed to load private key PEM files that include separate blocks for EC parameters (such as files output by OpenSSL’s EC key gen commands). This bug is now fixed. SERVER-3016

• **The `puppetserver ca generate` command errors because of the subject alternative name.** The `puppetserver ca generate` command no longer errors when `allow-subject-alt-names` is set to false. SERVER-3032

Puppet Server 6.16.0

Released June 2021 and shipped with Puppet Platform 6.23.0

New Features

• **The CA API accepts CRL updates.** You can now update your CRLs using the new API endpoint: PUT /puppet-ca/v1/certificate_revocation_list. This new endpoint accepts a list of CRL PEMs as a body, inserting updated copies of the applicable CRLs into the trust chain. The CA updates the matching CRLs saved on disk if the submitted ones have a higher CRL number than their counterparts. You can use this endpoint if your CRLs require frequent updates. Do not use the endpoint to update the CRL associated with the Puppet CA signing certificate (only earlier ones in the certificate chain) SERVER-2550

Enhancements

• **JRuby 9.2.17.0.** In this release, the JRuby version is updated to 9.2.17.0. SERVER-3007

Resolved issues

• **New apache HTTP client broke URL normalization.** A security update to the apache HTTP client introduced an unrelated change to URL normalization. This change affected any use of Puppet’s HTTP client within Puppet Server. In this release, the double slash in a URL path is no longer silently ignored by the HTTP client in Puppet Server. Instead, Puppet Server views it as a different URL and returns a 404. Going forward, remove leading double slashes from URLs. SERVER-3014

• **Environment endpoint failed to cache data if given valid etag.** Previously, if you used the environment and transport info endpoints, then you might have seen the cache bypassed—despite receiving a 304 Not Modified response. To work around this issue, users must submit a request to the environment_classes endpoint without the etag. This request triggers the correct caching behavior. Note that the console (the consumer of the environment_classes endpoint in PE) must always submit an etag for an environment if it has one. SERVER-3015

Puppet Server 6.15.3

Released 26 April 2021
Enhancements

• Puppet Server now adds an extension for subject-alternative-name (SAN) when it signs incoming certificate signing requests (CSR). The SAN extension contains the common name (CN) as a dns-name on the certificate. If the CSR comes with its own SAN extension, Puppet Server signs it and ensures that the SAN extension also includes the CSR's CN. SERVER-2338

Resolved issues

• The Jetty webserver now uses the local copy of the CRL from Puppet's SSL directory instead of the CA's copy. This fix makes it easier to set up compilers, which always have a disabled CA service and no CRL at the CA path. SERVER-2558
• Jetty has been updated to 9.4.40 to resolve security issues.

Puppet Server 6.15.1
Released 9 February 2021

Resolved issues

• Updated various dependencies to pick up security fixes.

Puppet Server 6.15.0
Released 20 January 2021

New features

• The puppetserver CA CLI now provides a `migrate` command to move the CA directory from the Puppet `confdir` to the puppetserver `confdir`. It leaves behind a symlink on the old CA location, pointing to the new location at `/etc/puppetlabs/puppetserver/ca`. The symlink provides backwards compatibility for tools still expecting the `cadir` to exist in the old location. In a future release, the `cadir` setting will be removed entirely. SERVER-2896

Puppet Server 6.14.1
Released 26 October 2020

Resolved issues

• The puppet-ca/v1/clean endpoint now logs the certname of each certificate it revokes. SERVER-2897

Puppet Server 6.14.0
Released 20 October 2020

New feature

• Added a new CA API endpoint — puppet-ca/v1/clean — that accepts a list of cert names to be revoked and deleted as a batch. SERVER-2859

Resolved issue

• Puppet Server's JRuby load path can now be used with `Dir.glob`. Notably, this re-enables installing gems with docs via `puppetserver gem`. SERVER-2763

Puppet Server 6.13.0
Released 25 August 2020

New features

• Puppet Server packages are now available for Ubuntu 20.04. SERVER-2828
• Added a new endpoint `/puppet-ca/v1/expirations` that returns the "not-after" date for each certificate in the CA bundle, as well as the "next-update" date of each CRL in the chain, keyed by common name. The endpoint requires authentication. SERVER-2551
Enhancement

• The /puppet-ca/v1/certificate_statuses endpoint now accepts a state parameter that will filter search results by the given certificate state. Accepted states are 'requested', 'signed', and 'revoked'. SERVER-2233

Resolved issue

• JRuby has been bumped to 9.2.13.0 for a security fix. SERVER-2853
• Re-enabled the ability to delete certificate signing requests via the CA API. SERVER-2795

Puppet Server 6.12.1

Released 14 July 2020

Enhancements

• Jolokia will no longer log at debug level by default, which avoids large stack traces for missing metrics. In order to re-enable debug output, set debug to true in metrics.conf and configure the logging to debug in logback.xml. TK-488
• The v2 metrics endpoint can now use trapperkeeper-authorization (tk-auth), which can be controlled from auth.conf (or from the authorization section of the trapperkeeper config). The v2 metrics endpoint is still restricted to localhost by default. If tk-auth is used to restrict access, you may override the default behavior in jolokia-access.xml. TK-489

Puppet Server 6.12.0

Released 3 June 2020

Resolved issue

• JRuby has been bumped to 9.2.11.1 again, with invokedynamic.yield set to false to resolve a stackoverflow error. SERVER-2793

Deprecation

• The v1 metrics endpoint, which was recently disabled by default, is now deprecated. Instead, use the v2 endpoint. TK-486

Puppet Server 6.11.1

Released 7 May 2020

Known issue

• JRuby has been rolled back to 9.2.8.0 while we investigate an intermittent problem where some requests that go through JRuby error repeatedly with StackOverflow exceptions. SERVER-2793.
• Downgrading JRuby reintroduced the sprintf bug marked fixed in 6.10.0, since its fix was tied to the JRuby update.

Puppet Server 6.11.0

Released 30 April 2020

New features

• The puppetserver ca CLI tool has been updated to version 1.7.0. It will now show any authorization extensions that exist when listing certificates or CSRs. SERVER-2591

Puppet Server 6.10.0

Released 14 April 2020

New features

• The GET /certificate_status endpoint now returns certificate or CSR's authorization extensions. SERVER-2718
• Puppet's ppRegCertExt arc has been extended with OID 1.3.6.1.4.1.34380.1.1.26 and the short name pp_owner. This OID is meant to help users in cloud environments. The short name will be displayed when using the puppetserver ca CLI tool.

Resolved issues
• Using a precision number to truncate a string in Puppet's sprintf function no longer interpolates extra characters. SERVER-2660.

Known issues
• An update to JRuby 9.2.11.1 has caused a change in defaults when installing gems with the puppetserver gem command. It attempts to install documentation by default, but this will not work. To avoid this bug, pass --no-document when installing gems. This is caused by an inability to use the classpath:/puppetserver-lib portion of the $LOAD_PATH as a parameter to Gem.list_files or Dir.glob, which Rdoc relies on to install documentation. SERVER-2758.

Puppet Server 6.9.2
Released 19 March 2020

Resolved issue
• To prevent information exposure as a result of CVE-2020-7943, the /metrics/v1 endpoints are disabled by default, and access to the /metrics/v2 endpoints are restricted to localhost.

Puppet Server 6.9.1
Released 10 March 2020

This release contains some minor test fixes.

Puppet Server 6.9.0
Released 18 February 2020

New features
• There is a new JRuby pool architecture that maintains a single a JRuby instance where requests to Puppet Server will run concurrently. You can toggle this behavior by setting jruby-puppet.multithreaded to true. In this mode, the server's memory footprint is significantly lighter as it no longer needs to run multiple JRuby instances. Note that this mode should be treated as an experimental feature. SERVER-2684

Puppet Server 6.8.0
Released 14 January 2020

New features
• When signing or generating certificates, you can now set the certificate time to live, either with a command line option or by specifying the key directly in the HTTP API. The time unit defaults to seconds, but you can specify a different time unit with any of time unit markers accepted in Puppet configuration.

The puppetserver ca sign and puppetserver ca generate commands accept a --ttl flag to set certificate time to live. This setting determines how long the resulting certificate is valid for.

Alternatively, you can set the time in the certificate-status API endpoint in the request body under the key cert_ttl. SERVER-2678

Resolved issues
• Puppet Server no longer issues HTTP 503 responses to agents older than Puppet 5.3, which can't react to these responses. This allows the max-queued-requests setting to be used safely with older agents. SERVER-2405

Puppet Server 6.7.2
Released 19 November 2019
This version contains minor security fixes.

**Puppet Server 6.7.1**
Released 15 October 2019

**Resolved issues**
- Puppet Server can no longer be configured to accept SSLv3 traffic. SERVER-2654

**Puppet Server 6.7.0**
Released 1 October 2019

**New feature**
- Puppet Server packages are now available for Debian 10. These packages require Java 11 to be installed, rather than Java 8. SERVER-2613

**Resolved issues**
- Puppet Server now synchronizes write access to the CRL, so that each revoke request updates the CRL in succession, instead of concurrently. This prevents corruption of the CRL due to competing requests.

**Puppet Server 6.6.0**
Released 17 September 2019

**New features**
- Puppet Server no longer hardcodes Java's egd parameter. Users may manage the value via JAVA_ARGS or JAVA_ARGS_CLI in the defaults file. SERVER-2602
- RedHat 7 FIPS mode packages are now available for puppetserver. SERVER-2555
- Puppet Server now lists plan content from your modules, just as it does task content. SERVER-2543
- You can now enable sending a list of all the Hiera keys looked up during compile to PuppetDB, via the jruby-puppet.track-lookups setting in puppetserver.conf. This is currently only used by CD4PE. SERVER-2538
- Added the /puppet-admin-api/v1/jruby-pool/thread-dump endpoint, which returns a thread dump of running JRuby instances, if jruby.management.enabled has been set to true in the JVM running Puppet Server. See Admin API: JRuby Pool for details. SERVER-2193
- Puppet Server now runs with JRuby 9.2.8.0. SERVER-2388
- The puppetserver ca import command now initializes an empty CRL for the intermediate CA if one is not provided in the crl-chain file. SERVER-2522

**Resolved issues**
- Puppet Server can now be reloaded and run with multiple JRuby instances when running under Java 11. This change affects the packaging of Puppet Server. If you are running Puppet Server from source, you must add facter.jar, provided by the puppet-agent package, to the classpath when starting Puppet Server with Java. SERVER-2423
- Puppet Server's CA can now handle keys in the PKCS#8 format, which is required when running in FIPS mode. SERVER-2019

**Puppet Server 6.5.0**
Released 22 July 2019

**New features**
- The default for the cipher-suites setting in the webserver section of webserver.conf has been updated. Previously, the defaults included 11 cipher suites, including 4 TLS_RSA_* cipher suites. Now the defaults include all cipher suites usable on a RHEL 7 FIPS-enabled server, our target platform for FIPS certification,
except for TLS_RSA_* ciphers. Additionally, Puppet Server emits warnings if any TLS_RSA_* ciphers are explicitly enabled in the cipher-suites setting.

To avoid potentially breaking clients that can use only TLS_RSA_* ciphers, the webserver.conf file now includes an explicit cipher-suites setting that adds the previously enabled TLS_RSA_* ciphers to the new implicit cipher-suites setting. This has three effects:

1. Older clients that require the TLS_RSA_* ciphers will continue to work.
2. Puppet Server generates warnings in the logs that the TLS_RSA_* ciphers are enabled.
3. Puppet Server generates warnings in the logs if ciphers enumerated in the cipher-suites setting are not available on that specific OS. These warnings can be safely silenced by editing the cipher-suites setting and removing the unavailable ciphers.

A future version of Puppet Server will remove the cipher-suites setting in webserver.conf. This will break any clients that still require the TLS_RSA_* ciphers.

In advance of this change, update any clients that still require the TLS_RSA_* ciphers to clients that can use more recent ciphers, and remove the cipher-suites setting in webserver.conf.

This update also removes the so-linger-seconds configuration setting. This setting is now ignored and a warning is issued if it is set. See Jetty’s so-linger-seconds for removal details.

See SERVER-2576 for further details.

- You can now specify a --certname flag with the puppetserver ca list command, which limits the output to information about the requested cert and logs an error if the requested cert does not exist in any form. SERVER-2589
- You can now specify a log level for the logs collected by the new catalog compilation endpoint during compilation. See the catalog endpoint docs for information. SERVER-2520
- In this release, performance in puppetserver commands is improved. Running puppetserver gem, puppetserver irb, and other Puppet Server CLI commands are 15-30 percent faster to start up. Service starting and reloading should see similar improvements, along with some marginal improvements to top-end performance, especially in environments with limited sources of entropy.
- Building Puppet Server outside our network is now slightly easier.
- Prior to this release, an unnecessary and deprecated version of Facter was shipped in the puppetserver package. This has been removed.
- Cert and CRL bundles no longer need to be in any specific order. By default, the leaf instances still come first, descending to the root, which are last. SERVER-2465

**Puppet Server 6.4.0**

Released 19 April 2019

**New features**

- This release adds a new API endpoint to /puppet/v3/environment_transports. This endpoint lists all of the available network transports from modules and is for use with the Agentless Catalog Executor. SERVER-2467

**Puppet Server 6.3.0**

Released 26 March 2019

**New features**

- Puppet Server has a new endpoint for catalog retrieval, allowing more options than the previous endpoint. This endpoint is controlled by tk-auth, and by default is not generally accessible. It is an API that integrators can use to provide functionality similar to puppet server --compile. This endpoint is intended for use by other Puppet services. SERVER-2434
Enhancements

- The CA's `certificate_status` endpoint now returns additional information for custom integration. SERVER-2370

Puppet Server 6.2.1

Released 20 February 2019.

This release contains resolved issues.

Resolved issues

- Updated bouncy-castle to 1.60 to fix security issues. SERVER-2431

Puppet Server 6.2.0

Released 23 January 2019.

This release contains new features and resolved issues.

New features

- The `puppetserver ca` tool now respects the `server_list` setting in `puppet.conf` for those users that have created their own high availability configuration using that feature. SERVER-2392
- The EZBake configs now allow you to specify `JAVA_ARGS_CLI`, which is used when using `puppetserver` subcommands to configure Java differently from what is needed for the service. This was used by the CLI before, but as an environment variable only, not as an EZBake config option. SERVER-2399

Resolved issues

- A dependency issue caused puppetserver 6.1.0 to fail with OpenJDK 11. This has been fixed and Puppet Server packages can now start under Java 11. SERVER-2404

Puppet Server 6.1.0

Released 18 December 2018

New features

- The CA service and the CA proxy service (in PE) now have their own entries in the status endpoint output and can be queried as "ca" and "ca-proxy" respectively. SERVER-2350
- Puppet Server now creates a default `ca.conf` file when installed, both in open source Puppet and Puppet Enterprise. CA settings such as `allow-subject-alt-names` should be configured in the `certificate-authority` section of this file. (SERVER-2372)
- The `puppetserver ca generate` command now has a flag `--ca-client` that will generate a certificate offline -- not using the CA API -- that is authorized to talk to that API. This can be used to regenerate the primary server's host cert, or create certs for distribution to other CA nodes that need administrative access to the CA, such as the ability to sign and revoke certs. This command should only be used while Puppet Server is offline, to avoid conflicts with cert serials. (SERVER-2320)
- The Puppet Server CA can now sign certificates with IP alt names in addition to DNS alt names (if signing certs with alt names is enabled). (SERVER-2267)

Enhancements

- Puppet Server 6.1.0 upgrades to JRuby 9.2.0.0. This version implements the Ruby 2.5 interface. It is backwards compatible, but will issue a warning for Ruby language features that have been deprecated. The major warning that users will see is `warning: constant ::Fixnum is deprecated`. Upgrading to this version of JRuby means that the Ruby interface has the same version as the Puppet agent. This version of JRuby is faster than previous versions under certain conditions. SERVER-2381
- Puppet Server now has experimental support for Java 11 for users that run from source or build their own packages. This has been tested with low level tests but does not work when installed from official packages. Consequently, we consider this support "experimental", with full support coming later in 2019 for the latest long term supported version of Java. SERVER-2315
• The puppetserver ca command now provides useful errors on connection issues and returns debugging information. SERVER-2317
• The puppetserver ca tool now prefers the server_list setting in puppet.conf for users that have created their own high availability configuration using this feature. SERVER-2392

Resolved issues
• The puppetserver ca command no longer has the wrong default value for the $server setting. Previously the puppetserver ca tool defaulted to $certname when connecting to the server, while the agent defaulted to puppet. The puppetserver ca tool now has the same default for $server as the agent. It will also honor the settings within the agent section of the puppet.conf file. SERVER-2354
• Jetty no longer reports its version. TK-473

Puppet Server 6.0.0
Released 18 September 2018

This Puppet Server release provides a new workflow and API for certificate issuance. By default, the server now generates a root and intermediate signing CA cert, rather than signing everything off the root. If you have an external certificate authority, you can generate an intermediate signing CA from it instead, and a new puppetserver ca subcommand puts everything into its proper place.

New features
• There is now a CLI command for setting up the certificate authority, called puppetserver ca. (SERVER-2172)
• For fresh installs, the Puppet primary server's cert is now authorized to connect to the certificate_status endpoint out of the box. This allows the new CA CLI tool to perform CA tasks via Puppet Server's CA API. (SERVER-2308) Note that upgrades will need to instead allow the primary server's cert for these endpoints.
• Puppet Server now has a setting called allow-authorization-extensions in the certificate-authority section of its config for enabling signing certs with authorization extensions. It is false by default. (SERVER-2290)
• Puppet Server now has a setting called allow-subject-alt-names in the certificate-authority section of its config for enabling signing certs with subject alternative names. It is false by default. (SERVER-2278)
• The puppetserver ca CLI now has an import subcommand for installing key and certificate files that you generate, for example, when you have an external root CA that you need Puppet Server's PKI to chain to. (SERVER-2261)
• We’ve added an infrastructure-only CRL in addition to the full CRL, that provides a list of certs that, when revoked, should be added to a separate CRL (useful for specifying special nodes in your infrastructure like compile servers). You can configure Whether this special CRL or the default CRL are distributed to agents. (SERVER-2231)
• Puppet Server now bundles its JRuby jar inside the main uberjar. This means the JRUBY_JAR setting is no longer valid, and a warning will be issued if it is set. (SERVER-2157)
• Puppet Server 6.0 uses JRuby 9K, which implements Ruby language version 2.3 Server-side gems that were installed manually with the puppetserver_gem command or using the puppetserver_gem package provider might need to be updated to work with JRuby 9K. Additionally, if ReservedCodeCache or MaxMetaspacesize parameters were set in JAVA_ARGS, they might need to be adjusted for JRuby 9K.
• The version of semantic_puppet has been updated in Puppet Server to ensure backwards compatibility in preparation for future major releases of Puppet Platform. (SERVER-2132)
• Puppet Server 6.0 now uses JRuby 9k. This implements version 2.3 of the Ruby language. (SERVER-2095)

Resolved issues
• We've made server-side fixes for fully supporting intermediate CA capability. With this, CRL chains will be persisted when revoking certs. SERVER-2205
Known issues

Ruby’s native methods for spawning processes cause a fork of the JVM on most Linux servers, which in a large production environment causes Out of Memory errors at the OS level. Puppet Server provides a lighter weight way of creating sub-processes with its built-in execution helper Puppet::Util::Execution.execute. Use Puppet::Util::Execution.execute when writing Ruby-based functions, custom report processors, Hiera backends and faces. When writing custom providers, use the commands helper to determine suitability.

Puppet Server known issues

For a list of all known issues, visit our Issue Tracker.

Access CA endpoint to update CRLs

Puppet Server 7.20 and 6.16.0 include the following new API endpoint: PUT /puppet-ca/v1/certificate_revocation_list. To access this endpoint, you must edit the configuration file at /etc/puppetlabs/puppetserver/conf.d/auth.conf and also update the endpoint's rule to be type regex instead of path.

Cipher updates in Puppet Server 6.5

Puppet Server 6.5 includes an upgrade to the latest release of Jetty’s 9.4 series. With this update, you may see "weak cipher" warnings about ciphers that were previously enabled by default. Puppet Server now defaults to stronger FIPS-compliant ciphers, but you must first remove the weak ciphers.

The ciphers previously enabled by default have not been changed, but are considered weak by the updated standards. Remove the weak ciphers by removing the cipher-suite configuration section from the webserver.conf. After you remove the cipher-suite, Puppet Server uses the FIPS-compliant ciphers instead. This release includes the weak ciphers for backward compatibility only.

The FIPS-compliant cipher suites, which are not considered weak, will be the default in a future version of Puppet. To maintain backwards compatibility, Puppet Server explicitly enables all cipher suites that were available as of Puppet Server 6.0. When you upgrade to Puppet Server 6.5.0, this affects you in in two ways:

1. The 6.5 package updates the webserver.conf file in Puppet Server's conf.d directory.
2. When Puppet Server starts or reloads, Jetty warns about weak cipher suites being enabled.

This update also removes the so-linger-seconds configuration setting. This setting is now ignored and a warning is issued if it is set. See Jetty’s so-linger-seconds for removal details.

Note: On some older operating systems, you might see additional warnings that newer cipher suites are unavailable. In this case, manage the contents of the webserver.cipher-suites configuration value to be those strong suites that available to you.

Server-side Ruby gems might need to be updated for upgrading from JRuby 1.7

When upgrading from Puppet Server 5 using JRuby 1.7 (9k was optional in those releases), Server-side gems that were installed manually with the puppetserver gem command or using the puppetserver_gem package provider might need to be updated to work with the newer JRuby. In most cases gems do not have APIs that break when upgrading from the Ruby versions implemented between JRuby 1.7 and JRuby 9k, so there might be no necessary updates. However, two notable exceptions are that the autosign gem should be 0.1.3 or later and yard-doc must be 0.9 or later.

Potential JAVA ARGS settings

If you're working outside of lab environment, increase ReservedCodeCache to 512m under normal load. If you're working with 6-12 JRuby instances (or a max-requests-per-instance value significantly less than 100k), run with a ReservedCodeCache of 1G. Twelve or more JRuby instances in a single server might require 2G or more.

Similar caveats regarding scaling ReservedCodeCache might apply if users are managing MaxMetaspace.
**tmp directory mounted noexec**

In some cases (especially for RHEL 7 installations) if the /tmp directory is mounted as noexec, Puppet Server may fail to run correctly, and you may see an error in the Puppet Server logs similar to the following:

```
Nov 12 17:46:12 fqdn.com java[56495]: Failed to load feature test for posix: can't find user for 0
Nov 12 17:46:12 fqdn.com java[56495]: Cannot run on Microsoft Windows without the win32-process, win32-dir and win32-service gems: Win32API only supported on win32
Nov 12 17:46:12 fqdn.com java[56495]: Puppet::Error: Cannot determine basic system flavour
```

This is caused by the fact that JRuby contains some embedded files which need to be copied somewhere on the filesystem before they can be executed (see this JRuby issue). To work around this issue, you can either mount the /tmp directory without noexec, or you can choose a different directory to use as the temporary directory for the Puppet Server process.

Either way, you'll need to set the permissions of the directory to 1777. This allows the Puppet Server JRuby process to write a file to /tmp and then execute it. If permissions are set incorrectly, you'll get a massive stack trace without much useful information in it.

To use a different temporary directory, you can set the following JVM property:

```
-Djava.io.tmpdir=/some/other/temporary/directory
```

When Puppet Server is installed from packages, add this property to the JAVAARGS and JAVAARGS_CLI variables defined in either /etc/sysconfig/puppetserver or /etc/default/puppetserver, depending on your distribution. Invocations of the gem, ruby, and irb subcommands use the updated JAVAARGS_CLI on their next invocation. The service will need to be restarted in order to re-read the JAVAARGS variable.

**Puppet Server Primary Server Fails to Connect to Load-Balanced Servers with Different SSL Certificates**

SERVER-207: Intermittent SSL connection failures have been seen when the Puppet Server primary server tries to make SSL requests to servers via the same virtual ip address. This has been seen when the servers present different certificates during the SSL handshake.

**Facter release notes**

These are the new features, resolved issues, and deprecations in this version of Facter.

**Facter 3.14.24**

Released August 2022 and shipped with Puppet Platform 6.28.0.

No release notes.

**Facter 3.14.23**

Released April 2022 and shipped with Puppet Platform 6.27.1.

No release notes.

**Facter 3.14.22**

Released January 2022 and shipped with Puppet Platform 6.26.0.

No release notes.
Facter 3.14.21
Released November 2021 and shipped with Puppet Platform 6.25.1.

Resolved issues
• Puppet 6.25.0 runs take over two minutes on non-EC2 VMs. This release fixes a regression where the AWS IMDSv2 endpoint was not called with the correct session and request timeouts, resulting in Facter using a default timeout of 2 minutes. FACT-3082

Facter 3.14.20
Released October 2021 and shipped with Puppet Platform 6.25.0.

New features
• DisplayVersion fact for Windows. This release adds a new fact called os.windows.display_version. This fact reads the version from the DisplayVersion registry key. FACT-3058
• AWS IDMSv2. This release ports the AWS Instance Metadata Service Version 2 (IMDSv2) functionality from Facter 4 to Facter 3. FACT-2904

Resolved issues
• Mismatched processor frequencies found on AIX. Previously, Facter added all the processors found in the ODM query, without checking their status. Now if the status is not available, Facter skips the processor. FACT-2955
• Windows 2022 detected as Windows 2019. The os.release.full and os.release.major facts now correctly detect Windows 2022 and output 2022, instead of 2019. FACT-3075

Facter 3.14.19
Released July 2021 and shipped with Puppet Platform 6.24.0.

This release includes minor maintenance changes. For the latest features, see the release notes for Facter 3.14.18.

Facter 3.14.18
Released June 2021 and shipped with Puppet Platform 6.23.0.

Enhancements
• Support for Mac OS X 11. This release adds support for Mac OS X 11 Big Sur. FACT-3017

Facter 3.14.17
Released 26 April 2021 and shipped with Puppet Platform 6.22.1.

Enhancements
• Azure identification fact. This release adds the cloud.provider fact for Azure identification on Linux and Windows platforms. FACT-1847
• Azure metadata fact. This release adds the az_metadata fact which provides information on Azure virtual machine instances. For more information, see the Microsoft Azure instance metadata documentation. FACT-1383
• Facter 3 API extended with resolve. This release backports a limited version of the Facter.resolve API to Facter 3. Note that the following options were not implemented: custom-dir, external-dir, no-block, no-cache, timing (not supported in Facter 3). FACT-2935
• Facter 3 support for Rocky Linux. This release adds support for the Rocky Linux operating system. Contributed by Puppet community member StackKorora. FACT-2957

© 2022 Puppet, Inc., a Perforce company
Resolved issues

- The puppet facts diff command logs error when apt module is installed. Previously, each time you executed require facter, the Facter and Puppet bin directories were added to PATH environment variable, even if they were already there. This is now fixed and require facter only adds Facter and Puppet bin directories to PATH if they are not there. FACT-2941

- Fix os.name detection of AlmaLinux. Facter 3 can now detect AlmaLinux. Contributed by Puppet community member cschug. FACT-2943

Facter 3.14.16

Resolved issues

- FACTER environmental facts overrides does not work with external facts. Previously, environment facts were overwritten by external facts when you ran the puppet fact command. Now the flow is set to facts -> external facts -> environment facts by default. FACT-2918

Facter 3.14.15
Released 20 January 2020 and shipped with Puppet Platform 6.20.0.

This release includes minor maintenance changes. For the latest features, see the release notes for Facter 3.14.14.

Facter 3.14.14
Released 20 October 2020 and shipped with Puppet Platform 6.28.0.

Resolved issues

Linux Mint 20 OS version bug. This release fixes an issue where Facter incorrectly reported the OS as Debian on Linux Mint 20 platforms. FACT-2759

Facter reports wrong Windows build. This release uses ntdll.dll instead of ntoskrnl.exe to get the Windows build number for the kernelrelease fact. FACT-2751

Facter 3.14.13
Released 25 August 2020 and shipped with Puppet Platform 6.18.0.

Resolved issues

Puppet lookup loads external facts on the initiating node. Previously, some Puppet default settings required Facter to retrieve data that loaded all external facts and slowed down Puppet. With this release, local external facts are not evaluated when performing a lookup command on a remote node. FACT-2666

Facter 3.14.12
Released 14 July 2020 and shipped with Puppet Platform 6.28.0.

Resolved issues

The facter -p command returns NUL character on stdout when called from nested ruby on Windows. This release fixes an issue where Facter executed a system command using back ticks when called from Ruby. FACT-2682

Facter 3.14.11
Released 3 June 2020 and shipped with Puppet Platform 6.16.0.

This release includes minor maintenance changes. For the latest features, see the release notes for Facter 3.14.10.
**Facter 3.14.10**
Released 30 April 2020 and shipped with Puppet Platform 6.15.0.

**New features**

`cached-custom-facts` has moved to a new section in `facter.conf`. This release moves `cached-custom-facts` to a new section in the `facter.conf` file, called `fact-groups`. FACT-2544

New ability to cache and block the output of custom facts. This release allows you to cache custom facts based on the `facter.conf` file. Custom facts defined in `facter.conf` under `Facts.cached-custom-facts` can be cached, depending on the `ttl` defined under `Facts.ttls`. FACT-1575

**Resolved issues**

AIX kernel resolver redirects `stderr` to `stdout` and breaks facts on OS errors. Previously, when the `oslevel -s` command is executed on AIX, redirects `stderr` to `/dev/null` were shown on `stderr` and the kernel fact broke. This is now fixed. FACT-2545

Not compatible with leatherman >= 1.0.0. This release adds a compile guard so that Facter can compile with older Leatherman versions. FACT-2537

No resolver for external facts file error for os-specific facts. Previously, when os-specific facts were not resolved, Puppet logged a warning. The release lowers the severity of the logged messages to debug. FACT-2489

Incorrect log format on non-executable external fact scripts. Previously, if an external fact script could not be executed, the Facter log did not indicate what happened and the error output format was incorrect. This is now fixed. FACT-2618

**Facter 3.14.9**

**Resolved issues**

- Facter no longer crashes if the user has a numeric hostname. (FACT-2346)
- Facter correctly displays the ssh host key fact, in the case the host key file does not contain a comment. (FACT-1833)

**Facter 3.14.8**
Released 18 February 2020 and shipped with Puppet Platform 6.13.0.

**New features**

- The ability to cache external facts using external facts filename as cache group. (FACT-2307)
- A new ssh fact on Windows — available when OpenSSH is present. (FACT-1934)

**Resolved issues**

- The `facter --puppet` command no longer throws a deprecation warning. (FACT-2260)

**Facter 3.14.7**

**New features**

- Facter command execution now accepts a Boolean parameter, `expand`. By default, Facter searches the command and expands it to absolute path. When `expand` is set to false, Facter verifies whether the command is a shell command and, if so, passes the command as is. FACT-2054

© 2022 Puppet, Inc., a Perforce company
Resolved issues

• Facter incorrectly reported disabled CPU cores as physical CPU cores. Now, Facter correctly reports physical and logical CPUs and ignores disabled CPUs. FACT-1824
• In previous versions, Facter could not always determine the primary network interface on Solaris, so it sometimes failed to return any valid interface. This is now fixed. FACT-2146
• In systems using Windows Remote Desktop Services (RDS), Facter returned an incorrect operating system fact. This was due to a Windows API deprecation that caused issues in mixed 32- and 64-bit application environments, such as RDS. FACT-2096
• Facter now takes the root-reserved space into account when reporting mountpoints. Contributed by Valia0906. FACT-2128

Facter 3.14.6
Released 19 November 2019 and shipped with Puppet Platform 6.11.0.

New features

• This release adds support for the fips_enabled fact on Windows. The check examines the contents of HKEY_LOCAL_MACHINE/System/CurrentControlSet/Control/Lsa/FipsAlgorithmPolicy/Enabled. If the returned value is 1, it means that FIPS mode is enabled. FACT-2065
• Facter can now return the new scope6 fact to display IPv6 address scope. FACT-2016

Resolved issues

• In previous releases, Facter did not report the cloud fact on Azure. This issue is now fixed. FACT-2004

Facter 3.14.5
Released 1 October 2019 and shipped with Puppet Platform 6.10.0.

Resolved issues

• Google Compute Engine’s internal metadata service is deprecating the v1beta1 endpoint sometime before the end of 2019. To prepare for this, Facter now uses the v1 endpoint instead. FACT-2018
• When Facter starts a mountpoint to get the size and available space, it causes mountpoints of type autofs to be automatically mounted, which is not the intended behavior. Automounts are now skipped by Facter when resolving mountpoints. FACT-1992

Facter 3.14.4
Released 17 September 2019 and shipped with Puppet Platform 6.9.0.

Resolved issues

This release makes the EC2 session timeout, in milliseconds, configurable via the EC2_SESSION_TIMEOUT environment variable. If the environment variable does not exist or is set to an invalid value, Facter defaults the variable to 5000 (5 seconds). FACT-1919

Facter 3.14.3
Released 21 August 2019 and shipped with Puppet Platform 6.8.0.

New features

This release adds an new AIX-specific fact, nim_type, which returns the type of the NIM configuration. This fact returns a value of:
• 'server'
• 'standalone'
• if '/etc/niminfo' is missing, no value is returned.
Resolved issues

- The previous version of Facter could not be compiled on the latest Archlinux version. This release fixes this issue by unpinning gem dependencies for Ruby spec tests so they work regardless of gem versions. and enabling detection of the optional UDEV library on Linux, which provides a serial number fact for disks. FACT-1968
- On Linux, if multiple mountpoints are mounted on the same directory, Facter showed only the first one retrieved from /etc/mtab. Now Facter shows the mountpoint that is a device or is the "tmpfs" type. FACT-1964
- Prior to this release, when you ran facter ipaddress6 on Windows servers, Facter returned the IPv6 with the interface identifier. This was caused by a Windows function that retrieves ipaddress6 together with the interface identifier. Now it correctly returns only the ipaddress6 without the interface id. This also fixes the network6 fact, which also returned network6 together with the interface identifier. FACT-1935
- The dmi fact did not return the chassis type. Facter now resolves the following DMI chassis descriptions for:
  - "25" - "Multi-system"
  - "26" - "CompactPCI"
  - "27" - "AdvancedTCA"
  - "28" - "Blade"
  - "29" - "Blade Enclosure"
  - "30" - "Tablet"
  - "31" - "Convertible"
  - "32" - "Detachable"
  FACT-1933

Facter 3.14.2
Released 23 July 2019 and shipped with Puppet Platform 6.7.0.

Resolved issues

- Prior to this release, Facter returned warnings if ip route show output was not in a key-value format. Because this format does not apply to all configurations, Facter no longer returns warnings about it. FACT-1916
- Previously, the mountpoint fact showed only temporary file systems and physical mounts. Now Facter returns mount points for all mounts on the system. FACT-1910

Facter 3.14.1
Released 1 July 2019 and shipped with Puppet Platform 6.6.0.

New features

- Facter now report disk serial numbers on Linux and FreeBSD FACT-1929
- This release adds a primary network interface check for FreeBSD. FACT-1926
- Previously, the mountpoint fact showed only temporary file systems and physical mounts. Now Facter returns mount points for all mounts on the system. FACT-1910
- This release adds support for Virtuzzo Linux facts. FACT-1888

Facter 3.14.0
Released 19 June 2019 and shipped with Puppet Platform 6.5.0

New features

In this release, Facter adds new facts for Windows version 10/2016+:
- ReleaseID: The four-digit Windows build version, in the form YYMM. On Windows10-1511-x86_64, the release ID is not displayed, as is not present in the registry.
- InstallationType: Differentiates Server, Server Core, Client (Desktop): Server|Server Core| Client.
• **ProductName**: Textual Product Name.

**Facter 3.13.1**
Released 26 March 2019 and shipped with Puppet Platform 6.4.0

**Resolved issues**
• This release fixes an issue where the dhcp fact failed on Red Hat 8 (beta). Because Red Hat 8 is a beta version, not all functions are verified to work correctly. FACT-1906

**Facter 3.13.0**
Released 20 February 2019 and shipped with Puppet Platform 6.3.0

**Resolved issues**
• Previously, Facter incorrectly reported operating system facts (such as os.name and os.release) on Ubuntu systems that did not have the lsb_release executable. Operating system facts are now resolved without relying on lsb_release. FACT-1899

**Facter 3.12.3**
Released 24 January 2019 and shipped with Puppet Platform 6.2.0.

**Resolved issues**
• Facter now returns the correct path to system32 on 64-bit systems where the sysnative folder has been created manually. FACT-1900

**Facter 3.12.2**
Released 18 December 2018 and shipped with Puppet Platform 6.1.0.

**Resolved issues**
• Previously, if you had multiple custom versions of a built-in fact, and only those with a weight of 0 could resolve, Facter used those zero-weighted values. Now, if only zero-weighted custom facts resolve, Facter uses built-in fact values instead. (FACT-1873)

**Facter 3.12.1**
Released 6.1.0 and shipped with Puppet Platform 18 December 2018.

**New features**
• Facter now correctly distinguishes between Windows Server 2016 and Windows Server 2019. FACT-1889

**Facter 3.12.0**
Released 18 September 2018 and shipped with Puppet Platform 6.0.0

**New features**
• Key type is now included as part of the facts for each SSH key. (FACT-1377)

**Resolved issues**
• Systems relying entirely on systemd-networkd for DHCP management do not use dhclient. This checks the DHCP leases directory of systemd-networkd (/run/systemd/netif/leases) in addition to the lease files of dhclient when attempting to identify DHCP servers. (FACT-1851)
• Facter no longer checks for missing dmidecode and does not report a warning when it’s missing on POWER Linux machines. (FACT-1765 and FACT-1763)
**Facter known issues**

These are the known issues in this version of Facter.

**Incorrect network output for VirtualBox or VPN network interfaces**

Running the `facter networking.interfaces` command in Facter 3 returns incorrect network output. This command currently works in Facter 4. [FACT-2951](#)

**Incorrect loopback interface output on AIX**

Running the `facter` command in Facter 3 returns incorrect network output for the loopback interface on AIX. These facts work correctly in Facter 4 and appear differently when running the `puppet facts diff` command in Facter 3. [FACT-2963](#)

**Incorrect mountpoints output on AIX**

Running the `facter` command in Facter 3 returns incorrect mountpoint output on AIX. These facts work correctly in Facter 4 and appear differently when running the `puppet facts diff` command in Facter 3. [FACT-2964](#)

**Facter 3 sets invalid values NVMe mountpoints**

An issue with Facter 3 causes invalid values for NVMe mountpoints. [FACT-3020](#)

**Facter 3 cannot not detect ldom facts**

Facter 3 cannot resolve the following facts on Solaris 11-SPARC: `ldom.domainchassis`, `ldom.domaincontrol`, `ldom_domainchassis`, and `ldom_domaincontrol`. [FACT-3010](#)

**Facter 3 cannot detect xvda disks on Amazon 6**

Facter 3 does not detect the type for `xvda` disks on Amazon 6 virtual machines. [FACT-3011](#)

**Facter 3 cannot resolve lsb facts if the `lsb_release` package is not installed**

Facter 3 cannot resolve the following facts if the `lsb_release` package is not installed on the box: `lsbdistrelease`, `lsbmajdistrelease`, `os.distro.codename`, `os.distro.id`, `os.distro.release.full`, `os.distro.release.major`. This is fixed in Facter 4. [FACT-3012](#)

**Facter 3 cannot detect scope6 fact**

The `scope6` fact was implemented in Facter 4, and is currently missing from Facter 3. [FACT-3013](#)

**Facter 3 cannot detect the VMware version**

For boxes running on ESXi, Facter 3 cannot detect the `hypervisors.vmware.version` fact. [FACT-3015](#)

**Facter 3 cannot detect IP v6 facts on Solaris**

Facter 3 cannot provide information on IP v6 facts on Solaris virtual machines. For a full list of facts, see [FACT-3016](#).

**Facter 3 cannot detect facts on Red Hat 7 AARCH**

Facter 3 cannot detect facts that are part of the partitions group. For a full list of facts, see [FACT-3019](#).
Facter 3 does not include Core(s) per socket and Thread(s) per core

Facter 3 does not include the Core(s) per socket and Thread(s) per core information in the processors fact. This information is available in Facter 4 only. FACT-2992

Inconsistencies with the --json flag

Running Facter 3 with the --json flag can produce a different outputs to Facter 4. For example, with facts that cannot be resolved when querying for fact names:

```bash
# facter 4
# bx facter non_existent --json
{
  "non_existent": null
}

# facter 3
# cfacter non_existent --json
{
  "non_existent": ""
}
```

This can also happen with external executable facts that do not produce any output, or have errors, creating entries in puppet facts diff. For example:

```bash
cat /etc/facter/facts.d/bad.sh
#!/bin/sh
echo bad=`abc`
puppet facts diff
Warning: Facter: external fact file "/etc/facter/facts.d/bad.sh" had output
on stderr: /etc/facter/facts.d/bad.sh: line 2: abc: command not found
Warning: Facter: external fact file "/etc/facter/facts.d/bad.sh" had output
on stderr: /etc/facter/facts.d/bad.sh: line 2: abc: command not found
{
  "bad": {
    "new_value": null,
    "old_value": ""
  }
}
```

FACT-3022

The macosx_productversion_major fact returns the wrong value for Big Sur

On macOS 11, the macosx_productversion_major fact incorrectly reports minor releases of Big Sur as major releases. For example, macOS 11.1 and 11.5 are treated as separate major releases, instead of minor releases of the macOS 11 series. FACT-3071

Getting started with Puppet Enterprise

Puppet Enterprise (PE) is automation software that helps you and your organization be productive and agile while managing your IT infrastructure.

PE is a commercial version of Puppet, our original open source product used by individuals managing smaller infrastructures. It has all the power and control of Puppet, plus a graphical user interface, orchestration services, role-based access control, reporting, and the capacity to manage thousands of nodes. PE incorporates other Puppet-related tools and products to deliver comprehensive configuration management capabilities.
There are two things you need to get started with PE: your content and the Puppet platform.

**Content**

You develop and store your automation content in a Git repository and upload it onto the Puppet platform. It consists of Puppet code, plan and task code, and Hiera data. You store content in a control repo, which contains bundles of code called modules and references to additional content from external sources, like the Puppet Forge — a repository of thousands of modules made by Puppet developers and the Puppet community.

**Puppet platform**

The Puppet platform includes the primary server, compilers, and agents. Use it to assign your desired state to managed systems, orchestrate ad-hoc automation tasks on managed and unmanaged systems, and get reports about configuration automation activity.

In this guide, you will learn how to install PE, add nodes to the console, set up your control repo, and run through an example task of managing webserver configurations — using either Apache to manage a *nix machine or IIS to manage a Windows machine.

Before you begin, check out our video for more information about how Puppet works.

- **Install PE** on page 158
  Installing Puppet Enterprise (PE) sets up a standard installation, which you can use to try out PE with up to 10 nodes or to manage up to 4,000 nodes.

- **Add nodes to the inventory** on page 160
  Your inventory is the list of nodes managed by Puppet. Add nodes with agents, agentless nodes that connect over SSH or WinRM, or add network devices like network switches and firewalls. Agent nodes help keep your infrastructure in your desired state. Agentless nodes do not have an agent installed, but can do things like run tasks and plans.

- **Add code and set up Code Manager** on page 161
  Set up your control repo, create a Puppetfile, and configure Code Manager so you can start adding content to your Puppet Enterprise (PE) environments.

- **Manage Apache configuration on *nix targets** on page 164
  Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

- **Manage IIS configuration on Windows targets** on page 170
  Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

- **Next steps** on page 177
  Now that you have set up some basic automated configuration management with PE, here are some things to do next.

**Install PE**

Installing Puppet Enterprise (PE) sets up a standard installation, which you can use to try out PE with up to 10 nodes or to manage up to 4,000 nodes.

Ensure your system capacity can manage a PE installation by reviewing the Hardware requirements for standard installations on page 182.

A standard PE installation consists of these components installed on a single node:

- The primary server: The central hub of activity, where Puppet code is compiled to create agent catalogs, and where SSL certificates are verified and signed.
• The console: The graphical web interface, which features configuration and reporting tools.
• PuppetDB: The data store for data generated throughout your Puppet infrastructure.

**Important:** The primary server runs only on *nix machines. Windows machines can run as Puppet agents, and you can manage them with your *nix primary server. If you want to operate a *nix primary server remotely from a Windows machine, before installing PE, configure an SSH client, such as PuTTY, with the hostname or IP address and port of the *nix machine that you want to install as your primary server. When you open an SSH session to install the *nix primary server, log in as the root user or use `sudo`.

### Install PE

Installation uses default settings to install PE infrastructure components on a single node. After installing, you can scale or customize your installation as needed.

**Important:** Perform these steps on your target primary server logged in as root. If you're installing on a system that doesn't allow root login, use the command `sudo su -` to switch to the root user.

1. **Download** the tarball appropriate to your operating system and architecture.

   **Tip:** To download packages from the command line, run `wget --content-disposition <URL>` or `curl -JLO <URL>`, using the URL for the tarball you want to download.

2. **Unpack the installation tarball:**

   ```
tar -xf <TARBALL_FILENAME>
   ```

3. **From the installer directory,** run the installer and follow the CLI instructions to complete your installation:

   ```
./puppet-enterprise-installer
   ```

4. **Optional:** Restart the shell in order to use client tool commands.

### Log in to the PE console

The console is a graphical interface where you can manage your infrastructure without relying on the command line.

To log in for the first time:

1. **Open the console** by entering the URL `<PRIMARY_HOSTNAME>` into your browser, where hostname is your primary server's trusted certificate name.

   **Note:** You'll receive a browser warning about an untrusted certificate because you were the signing authority for the console’s certificate, and your Puppet Enterprise deployment is not known to your browser as a valid signing authority. Ignore the warning and accept the certificate.

2. On the console login page, log in with the username `admin` and the password you created when installing. Keep track of this login because you need it later.

Next, check the status of your primary server.

### Check the status of your primary server

You can run a task to check the status of your primary server in the console.

A task is a single action that allows you to do ad-hoc things like upgrade packages and restart services on target machines. Puppet Enterprise (PE) comes with a few tasks installed, such as `package`, `service`, and `puppet_conf`, and you can download more tasks from the Forge or write your own.

1. **In the console,** in the **Orchestration** section, click **Tasks**.
2. **Click Run a task** in the upper right corner of the **Tasks** page.
3. **In the Task field,** select `service` because you are checking the status of the primary server service.
4. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

5. Under **Task parameters**, parameters and values for the task. The *service* task has two required parameters. For **action**, choose **status**. For **name**, enter **puppet**.

6. Under **Select targets**, select **Node list**.
   
   a) In the **Inventory nodes** field, add the hostname of your primary server and select it.

7. Click **Run task** or **Schedule job**.

   Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

   **Tip:** Filter run results by task name to find specific task runs.

View the task status and output in the **Jobs** page when it is finished running.

Your primary server's status should be “running” and “enabled”. To learn more about tasks, including how to install them from the Forge or how to write your own, visit the *Installing tasks* and *Writing tasks* sections of the docs.

Next, use the console to add nodes to the inventory on page 160.

---

**Add nodes to the inventory**

Your inventory is the list of nodes managed by Puppet. Add nodes with agents, agentless nodes that connect over SSH or WinRM, or add network devices like network switches and firewalls. Agent nodes help keep your infrastructure in your desired state. Agentless nodes do not have an agent installed, but can do things like run tasks and plans.

**Add agent nodes**

Use the console to add agent nodes to your inventory. Agents help with configuration management by periodically correcting changes to resources and reporting information to the primary server about your infrastructure. Note that adding an agent node installs the agent on the node.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Install agents**.
3. Select the transport method. This connection is used to remotely install the agent on the target node.
   
   • **SSH** for *nix targets
   • **WinRM** for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click **Add nodes**.

   **Tip:** Click **Installation job started** to view the job details for the task.

   Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

**Add agentless nodes**

Add nodes that will not or cannot have an agent installed on them. Agentless automation allows you to do things like update a package or restart a server on demand for node targets that don't have software installed.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect over SSH or WinRM**.
3. Select a transport method.
   - SSH for *nix targets
   - WinRM for Windows targets

4. Enter target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.

5. Optional: Select additional target options. For example, to add a target port number, select Target Port from the drop-down list, enter the number, and click Add. For details, see Transport configuration options on page 455.

6. Click Add nodes.

After adding the nodes to the inventory, they are added to PuppetDB, and you can view them from the Nodes page. Nodes in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory node's connection options, or to remove the node from inventory, go to the Connections tab on the node's details page.

Managing certificate signing requests in the console

A certificate signing request appears in the console on the Certificates page in the Unsigned certificates tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

- To manage requests individually, click Accept or Reject.
- To manage the entire list of requests, click Accept All or Reject All. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

Related information

Running Puppet on demand on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

Add code and set up Code Manager

Set up your control repo, create a Puppetfile, and configure Code Manager so you can start adding content to your Puppet Enterprise (PE) environments.

The control repo is where you store your code. Code in your control repo is usually bundled in modules.

The Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Code Manager automates the management and deployment of your Puppet code. PE doesn't require Code Manager, but it is helpful for ensuring Puppet syncs code to your primary server and all your servers run new code at the same time.

Create a control repository from the Puppet template

To create a control repository (or control repo) that has the recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

The Puppet control repo template contains the necessary files to configure a functioning code management control repo plus helpful Puppet code examples, including:

- Basic code examples for setting up roles and profiles.
- A Puppetfile that references modules to manage content in your environments.
- An example Hiera configuration file and hieradata directory.
- A config_version script that tells you which version of code from your control repo was applied to your agents.
An environment.conf file that implements the config_version script and a site-modules directory for roles, profiles, and custom modules.

In situations where you can't access the internet, or where organizational security policies prevent downloading modules from the Forge, you can Create an empty control repo on page 766 and add the necessary files to it.

To use the template, you must set up a private SSH key, copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

**Important:** The following steps assume you are using GitHub Enterprise with SSH. For more information and instructions for other version control hosts, such as GitLab or Bitbucket, go to the Puppet control-repo template README.

1. To allow access to the control repo, generate a private SSH key without a password:
   a) To generate the key pair, run:
   ```bash
   ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
   ```
   b) To allow the pe-puppet user to access the key, run:
   ```bash
   puppet infrastructure configure
   ```
   Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.
   c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as GitHub, BitBucket Server, or GitLab).

   **Important:** Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

2. In your Git user account or organization, create a repository named control-repo, and make sure a README is not automatically generated when you create the repo. Take note of the repo's SSH URL.

   **Important:** Do not use an existing repo. The template requires a new, empty repo named control-repo.

3. If you haven't already installed Git, run `yum install git`.
4. To clone the Puppet control-repo template, run:
   ```bash
   git clone https://github.com/puppetlabs/control-repo.git
   ```
5. Change to the control-repo directory: `cd control-repo`
6. Remove the template repo as the origin: `git remote remove origin`
7. Set your control repo as the origin: `git remote add origin <SSH_URL_FOR_YOUR_CONTROL_REPO>`
8. Push the contents of the production branch of the cloned control repo to your remote control repo: `git push origin production`

You now have a control repository based on the Puppet control-repo template. After configuring Code Manager, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, Code Manager detects and deploys your infrastructure changes.

By using the control-repo template, you now also have a Puppetfile to which you can add and manage content, like module code.
Related information

Managing environment content with a Puppetfile on page 768
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Managing code with Code Manager on page 774
Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

Add an environment on page 768
Create new environments by creating branches based on your control repo's production branch.

Configure Code Manager

Code Manager stages, commits, and synchronizes your code, automatically managing your environments and modules when you make changes.

Enable Code Manager
Set parameters in the console to enable Code Manager.

Before you begin
See Configure settings using the console on page 286 for more details about setting parameters in the console.

1. In the console, click Node groups, and set the following parameters in the puppet_enterprise::profile::master class in the PE Master node group.
   - Set code_manager_auto_configure to true.
   - Set r10k_remote to your control repository's location. Enter a string that is a valid URL for your Git control repository, such as git@<YOUR.GIT.SERVER.COM>:puppet/control.git.
   - For r10k_private_key, enter a string specifying the path to the SSH private key that permits the pe-puppet user to access your Git repositories. This file must be:
     - Located on the primary server.
     - Owned by the pe-puppet user.
     - Located in a directory that the pe-puppet user has permission to view. We recommend: /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519

2. Click Commit.

3. On the command line, run puppet job run --nodes <NODE NAME> where <NODE NAME> is the name of your primary server.
   
   For example:
   
   puppet job run --nodes small-doubt.delivery.puppetlabs.net

   **Note:** r10k is a code management tool that allows you to manage your environment configurations — production, testing, and development — in a source control repository.

Next, set up authentication for Code Manager.

Set up authentication for Code Manager
To securely deploy environments, Code Manager needs an authentication token for both authentication and authorization.

To generate a token for Code Manager:

1. Assign a user to the deployment role.
2. In the console, create a deployment user.

   **Tip:** Create a dedicated deployment user for Code Manager use.

3. Add the deployment user to the **Code Deployers** role.

   **Note:** This role is automatically created on install, with default permissions for code deployment and token lifetime management.

4. Create a password by clicking **Generate Password**.

5. **Request an authentication token for deployments** on page 164

---

**Related information**

**Configure puppet-access** on page 347

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. Ensure `puppet-access` is configured correctly before using it to generate authentication tokens.

**Request an authentication token for deployments**

To enable secure deployment of your code, request an authentication token for the deployment user.

By default, authentication tokens have a one-hour lifetime. With the `Override default expiry` permission set, you can change the lifetime of the token to a duration better suited for a long-running, automated process.

1. From the command line on the primary server, run `puppet-access login --lifetime 180d`. This command both requests the token and sets the token lifetime to 180 days.

   **Tip:** You can add flags to the request to specify additional settings such as the token file's location or the URL for your RBAC API. See **Configuration file settings for puppet-access**.

2. Enter the user name and password of the deployment user when prompted.

The generated token is stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. To view the token, run `puppet-access show`.

**Related information**

**Setting a token-specific lifetime** on page 351

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

**Generate a token for use by a service** on page 351

If you need to generate a token for use by a service and the token doesn’t need to be saved, use the `--print` option.

---

**Deploy your code**

Use the command line to trigger Code Manager after making changes to your Puppetfile.

When you make changes to your Puppetfile, like adding a new module or creating a repo, you must deploy your code before Code Manager can recognize or start managing the content.

SSH into your primary server and run `puppet-code deploy --all --wait`.

You have deployed code to all environments. The `--wait` flag returns results after the deployment is finished. Use the command `puppet-code deploy <ENVIRONMENT>` to deploy code to only a specific environment. You can also deploy code using a **webhook** or **custom scripts**.

---

**Manage Apache configuration on *nix targets**

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.
Here we demonstrate how to distribute Apache services to a node group by installing the `apache` module, creating a node group for the Apache nodes, creating profiles to specify Apache and webserver configurations, bundling the profiles into a role, and assigning the role to the node group. Once this is done, PE distributes the role to the individual nodes and ensures the individual nodes have the Apache service and the desired configurations.

- **Install the apache module** on page 165
  Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the `puppetlabs/apache` module, which automates installing, configuring, and managing Apache services.

- **Set up Apache node groups** on page 166
  Tell Puppet Enterprise (PE) your desired infrastructure configuration by creating *node groups* to categorize related nodes based on their function. Before you begin, decide which of your inventory nodes you want to have Apache services.

- **Organize webserver configurations with roles and profiles** on page 167
  The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

### Install the apache module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the `puppetlabs/apache` module, which automates installing, configuring, and managing Apache services.

#### Before you begin

**Install PE** on page 158 and at least one *nix agent node before installing the `apache` module.

**Tip:** You can write your own modules or download pre-built modules from the Forge. While you can use any module on the Forge, PE customers can take advantage of supported modules. These modules are designed to facilitate common services, and they are tested and maintained by Puppet.

1. Go to the [Apache module](https://forge.puppet.com) page on the Forge.
2. Select r10k or Code Manager as the *Installation Method*, and follow the instructions to add the module declaration to your Puppetfile.

   By default, Code Manager installs the latest version and disables automatic updates; however, you can specify options to install a different version or keep the module current with the latest version. To automatically update the module when a new version is released, specify `:latest` (such as `mod 'puppetlabs/apache', :latest`). To install a specific version of the module and prevent automatic updates, specify the version number as a string (such as `mod 'puppetlabs/apache', '5.4.0'`).

3. Make sure your Puppetfile includes module declarations for the `puppetlabs/stdlib` and `puppetlabs(concat)` modules, which are dependencies of the `apache` module. Dependencies for each module are listed on the Dependencies tab on the module’s Forge page, and you can specify the desired version in the same way you did for the primary module.

   For example, this code installs version 5.4.0 of the `apache` module, installs the module's dependencies, and prevents automatic updates (due to specified version numbers):

   ```
   mod 'puppetlabs/apache', '5.4.0'
   mod 'puppetlabs/stdlib', '4.13.1'
   mod 'puppetlabs/concat', '2.2.1'
   ```

4. SSH into your primary server and run `puppet-code deploy --all` to deploy code.

You installed the `apache` module. Installing a module makes it available in PE so you can use it to manage nodes.
After installing a module, you can run tasks included in the module from the PE console. The `apache` module contains a task that allows you to perform Apache service functions. To view or run the `apache` task in the PE console, go to the Run section, under Task, and enter `apache` in the Task field.

If the `apache` task doesn't appear, try refreshing the console in the browser. If it still does not appear, check that the `puppetlabs/apache` module is in your Puppetfile and try again.

To continue managing Apache configuration on *nix targets, Set up Apache node groups on page 166.

Related information
Managing environment content with a Puppetfile on page 768
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Set up Apache node groups
Tell Puppet Enterprise (PE) your desired infrastructure configuration by creating node groups to categorize related nodes based on their function. Before you begin, decide which of your inventory nodes you want to have Apache services.

Important: Setting up a node group does not install software or configure the individual nodes in the node group. A node group is a categorical designation for related nodes (in this case, nodes running the Apache module), and you use the node group to manage nodes in bulk. After creating a node group, you must apply a role to the group so the nodes have some specifications to inherit.

Create your classification group
A classification node group is a parent group for other node groups that contain classification data. You only need to set up the classification node group once.

The classification group distinguishes classification node groups from environment node groups.

1. In the console, click Node groups, and click Add group.
2. Specify options for the new node group:
   - Parent name: All Nodes
   - Group name: All Classification
   - Environment: production
   - Environment group: Do not select
3. Click Add.

You created the classification group.

Create a node group and add it as a child of the classification group.

Create your apache node group
After creating the parent classification group, make the `apache` node group as a child of the classification group.

1. In the console, click Node groups, and click Add group.
2. Specify options for the `apache` node group:
   - Parent name: All Classification
   - Group name: apache
   - Environment: production
   - Environment group: Do not select
3. Click Add.

You have created a node group to categorize the nodes you want to have Apache services.

Identify nodes from your inventory that you want to run Apache on and add them to the `apache` node group.
Add nodes to the *apache* node group

Once you determine which nodes from your inventory belong in your *apache* node group, pin the nodes to the group. Pinning adds nodes to a group one at a time. If you have a lot of nodes, you can create rules in the console to dynamically add nodes to a node group.

**Best practices for classifying node groups** on page 460 can help you decide which nodes belong in your *apache* node group.

1. In the console, click **Node groups** and select the *apache* node group.
2. On the **Rules** tab, under **Certname**, enter the certname of a node you want to add to the *apache* node group.
3. Click **Pin node**.
4. Repeat to add more nodes to the *apache* group.
5. When you are done pinning nodes, commit changes.

You have added your *apache* nodes to the *apache* node group.

Run Puppet.

**Related information**

**Making changes to node groups** on page 468

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

**Grouping and classifying nodes** on page 460

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

**Run Puppet on your Apache nodes**

Run Puppet in the console to enforce your desired state on the *apache* node group you created.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Under **Run options**, do not select anything.
3. From the list of target types, select **Node group**.
4. In the **Chose a node group box**, search for the *apache* node group and click **Select**.
5. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun the job on all nodes or only on the nodes that failed during the initial run.

After running Puppet on a node group, the nodes in the group automatically check in with PE every 30 minutes to confirm that their configuration matches what you've specified for this group. PE corrects the configuration if it doesn't match.

You must create roles and profiles to define specific applications, services, and configurations you want the nodes in this group to receive, as explained in **Organize webserver configurations with roles and profiles** on page 167.

**Important:** Until you assign a role to a node group, the group has no defined configuration for the nodes to inherit.

**Organize webserver configurations with roles and profiles**

The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

Roles and profiles allow you to select relevant pieces of code from modules and bundle them together to create your own custom set of code for managing things. *Profiles* are the individual bundles of code. *Roles* gather profiles together so you can assign them to nodes. This allows you to efficiently organize your Puppet code.

To illustrate roles and profiles, these steps demonstrate how to:

- Define a profile that configures virtual webhost (vhost) to serve the *example.com* website with a firewall rule.
- Create a role to contain the profile.
- Assign the role to the *apache* node group.
This creates a base structure where, if you had additional websites to serve, you would create additional profiles for those sites. When you have multiple profiles, you can combine profiles within roles or create unique roles for each profile.

Because this example adds a firewall rule, make sure you add the puppetlabs/firewall module to your Puppetfile, following the same process you used to Install the apache module on page 165. Remember to add the firewall modules dependencies (puppetlabs/stdlib), such as:

```plaintext
mod 'puppetlabs/firewall', '2.3.2'
mod 'puppetlabs/stdlib', '4.0.0'
```

Related information

The roles and profiles method on page 502

The roles and profiles method is the most reliable way to build reusable, configurable, and refactoring system configurations.

Set up your prerequisites

Before writing content for roles and profiles, you need to create modules to store them in.

1. Create one module for profile and one for role directly in your control repo. Do not put them in your Puppetfile.
2. Make a new directory in the control repo named site. For example, /etc/puppetlabs/code/environments/production/site.
3. Add site to the modulepath in the environment.conf file. The modulepath is the place where Puppet looks for module information. For example: modulepath = site:modules:$basemodulepath.
4. Put the role and profile modules in the site directory.

Write a profile for your Apache vhost

Write a webserver profile that includes rules for your Apache vhost and firewall.

Before you begin

Make sure you have:

- Installed the puppetlabs/apache module, the puppetlabs/firewall module, and their dependencies from the Forge.
- Created the role and profile modules, as explained in Set up your prerequisites.

Tip: We recommend writing your code in a code editor, such as VSCode, and then pushing to your Git server. There is a Puppet VSCode extension that supports syntax highlighting of the Puppet language.

1. In the profile module, create the following directories and .pp file:

   - manifests/
     - webserver/
       - example.pp
2. Paste this Puppet code into the example.pp file:

```puppet
# /etc/puppetlabs/code/environments/production/site/profile/manifests/webserver/example.pp
class profile::webserver::example (
    String $content      = "Hello from vhost\n",
    Array[String] $ports = ['80']
    Array[String] $ips   = ['127.0.0.1','127.0.0.2'],
) {
    class { 'apache':
        default_vhost => false,
    }
    apache::vhost { 'example.com':
        port => $ports,
        ip => $ips,
        ip_based => true,
        docroot => '/var/www/html',
    } file { '/var/www/html/index.html':
        ensure => file,
        content => $content,
    } firewall { '100 allow http and https access':
        dport => $ports,
        proto => tcp,
        action => accept
    }
}
```

This profile applies custom rules for the apache::vhost class that include arrays of $ports and $ips. The code uses file to ensure your vhost's main page has content. Finally, there is a firewall rule that only allows traffic from the ports set in the $ports array.

You can add your own code to the profile as needed. For more information, go to these Forge pages:

- Apache module README
- Apache module Reference
- Firewall module README
- Firewall module Reference

**Set data for the profile**

Hiera is a configuration method that allows you to set defaults in your code or override defaults (in certain circumstances). Use it to refine profile data.

Suppose you want to use the custom fact stage to represent the deployment stage of the node, which can be dev, test, or prod. For this example, use dev and prod.

With Hiera structured data, you can set up a four-layer hierarchy consisting of:

- console_data for data defined in the console.
- nodes/%{trusted.certname} for per-node overrides.
- stage/%{facts.stage} for setting stage-specific data.
- common for global fallback data.

This structure lets you tune the settings for ports and IPs in each stage.

For example, to make webservers in the development environment have a custom message and use port 8080, you'd create a data file with the following name, location, and code content:

```yaml
# cat /etc/puppetlabs/code/environments/production/data/stage/dev.yaml
```
---

profile::webserver::example::content: "Hello from dev\n"
profile::webserver::example::ports:
  - '8080'

You’d use this code to make web servers in the production environment listen to all interfaces:

```yaml
# cat /etc/puppetlabs/code/environments/production/data/stage/prod.yaml
---
profile::webserver::example::ips:
  - '0.0.0.0'
  - '::'
```

This is a brief introduction to what you can do with structured data in Hiera. To learn more about setting up hierarchical data, see Getting started with Hiera.

**Write a role for your Apache webserver**

Roles contain sets of profiles. To write roles, think about the machines you’re managing and decide what else they need in addition to the `webserver` profile.

This example shows how to write a role by combining profiles. In this example, assume you want all nodes in your `apache` node group to use the `webserver` profile you just wrote, and that your organization assigns all machines (including workstations) a profile called `profile::base` that manages basic policies and uses some conditional logic to include operating-system-specific configuration.

1. In your control repo, open the `.pp` file for the `role` module. If it doesn’t exist, create the necessary directories and file, such as:

```
/etc/puppetlabs/code/environments/production/site/role/manifests/exampleserver.pp
```

2. Write a role that includes both the `base` profile and your `webserver` profile:

   ```puppet
   class role::exampleserver {
     include profile::base
     include profile::webserver
   }
   ```

3. You can add more profiles to this role, or create additional roles with more profile configurations based on your needs.

**Assign the role to nodes**

Assign the `exampleserver` role to the node group containing the nodes that you want to have the Apache vhost configuration you wrote in the `webserver::example` profile.

For this example, assume you want to add `role::exampleserver` to all nodes in the `apache` node group.

1. In the console, click **Node groups** and select the `apache` node group.
2. On the **Classes** tab, select `role::exampleserver` and click **Add class**.
3. Commit the change.

Now, the `apache` node group manages your Apache vhost based on the rules you wrote in your `webserver` profile. When the nodes check in with PE, PE distributes the role (and the contained profiles) to the individual nodes and ensures the individual nodes have the Apache service and the desired configurations.

**Manage IIS configuration on Windows targets**

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy
relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

Here we demonstrate how to distribute IIS services to a node group by installing the iis module, creating a node group for the IIS nodes, creating profiles to specify IIS and webserver configurations, bundling the profiles into a role, and assigning the role to the node group. Once this is done, PE distributes the role to the individual nodes and ensures the individual nodes have the IIS service and the desired configurations.

- **Install the iis module** on page 171
  Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the puppetlabs/iis module, which automates installing, configuring, and managing IIS services.

- **Set up IIS node groups** on page 172
  Tell Puppet Enterprise (PE) your desired infrastructure configuration by grouping and classifying nodes based on their function. Before you begin, decide which of your inventory nodes you want to have IIS services.

- **Organize webserver configurations with roles and profiles** on page 173
  The roles and profiles method is a reliable way to build reusable, configurable, and refactorable system configurations.

### Install the iis module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the puppetlabs/iis module, which automates installing, configuring, and managing IIS services.

#### Before you begin

- **Install PE** on page 158 and at least one Windows agent node before installing the iis module.

- **Tip:** You can write your own modules or download pre-built modules from the Forge. While you can use any module on the Forge, PE customers can take advantage of supported modules. These modules are designed to facilitate common services, and they are tested and maintained by Puppet.

1. Go to the IIS module page on the Forge.
2. Select **r10k** or Code Manager as the **Installation Method**, and follow the instructions to add the module declaration to your Puppetfile.

   By default, Code Manager installs the latest version and disables automatic updates; however, you can specify options to install a different version or keep the module current with the latest version. To automatically update the module when a new version is released, specify :latest (such as `mod 'puppetlabs/iis', :latest`). To install a specific version of the module and prevent automatic updates, specify the version number as a string (such as `mod 'puppetlabs/iis, '7.0.0'`).

3. Make sure your Puppetfile includes module declarations for the puppetlabs/pwshlib module, which is a dependency of the iis module. Dependencies for each module are listed on the Dependencies tab on the module’s Forge page, and you can specify the desired version in the same way you did for the primary module. For example, this code installs version 7.0.0 of the iis module, installs the module's dependency, and prevents automatic updates (due to specified version numbers):

   ```
   mod 'puppetlabs/iis, '7.0.0'
   mod 'puppetlabs/pwshlib', '0.4.0'
   ```

4. SSH into your primary server and deploy code running the puppet-code deploy --all command.

   You installed the iis module. Installing a module makes it available in PE so you can use it to manage nodes.

To continue managing IIS configuration on Windows targets, **Set up IIS node groups** on page 172.
Related information

Managing environment content with a Puppetfile on page 768
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Set up IIS node groups

Tell Puppet Enterprise (PE) your desired infrastructure configuration by grouping and classifying nodes based on their function. Before you begin, decide which of your inventory nodes you want to have IIS services.

**Important:** Setting up a node group does not install software or configure the individual nodes in the node group. A node group is a categorical designation for related nodes (in this case, nodes running the IIS module), and you use the node group to manage nodes in bulk. After creating a node group, you must apply a role to the group so the nodes have some specifications to inherit.

Create your classification group
A classification node group is a parent group for other node groups that contain classification data. You only need to set up the classification node group once.

The classification group distinguishes classification node groups from environment node groups.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group:
   - **Parent name**: All Nodes
   - **Group name**: All Classification
   - **Environment**: production
   - **Environment group**: Do not select
3. Click **Add**.

You created the classification group.

Create a node group and add it as a child of the classification group.

**Create your iis node group**
After creating the parent classification group, make the iis node group as a child of the classification group.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the iis node group:
   - **Parent name**: All Classification
   - **Group name**: iis
   - **Environment**: production
   - **Environment group**: Do not select
3. Click **Add**.

You have created a node group to categorize the nodes you want to have IIS services. Next, determine which nodes from your inventory you want to run IIS on and add them to the iis node group.

Identify nodes from your inventory that you want to run IIS on and add them to the iis node group.

**Add nodes to the iis node group**
Once you determine which nodes from your inventory belong in your iis node group, pin the nodes to the group. Pinning adds nodes to a group one at a time. If you have a lot of nodes, you can create rules in the console to dynamically add nodes to a node group.

**Best practices for classifying node groups** on page 460 can help you decide which nodes belong in your iis node group.

1. In the console, click **Node groups** and select the iis node group.
2. On the **Rules** tab, under **Certname**, enter the certname of a node you want to add to the **iis** node group.
3. Click **Pin node**.
4. Repeat to add more nodes to the **iis** group.
5. When you are done pinning nodes, commit changes.

You have added your **iis** nodes to the **iis** node group.

Run Puppet.

**Related information**

*Making changes to node groups* on page 468
You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

*Grouping and classifying nodes* on page 460
Configure nodes by assigning classes, parameters, and variables to them. This is called **classification**.

**Run Puppet on your IIS nodes**

Run Puppet in the console to enforce your desired state on the **iis** node group you created.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Under **Run options**, do not select anything.
3. From the list of target types, select **Node group**.
4. In the **Chose a node group** box, search for the **iis** node group and click **Select**.
5. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun the it on all nodes or only on the nodes that failed during the initial run.

After running Puppet on a node group, the nodes in the group automatically check in with PE every 30 minutes to confirm that their configuration matches what you've specified for this group. PE corrects the configuration if it doesn't match.

You must create roles and profiles to define specific applications, services, and configurations you want the nodes in this group to receive, as explained in *Organize webserver configurations with roles and profiles* on page 173.

**Important:** Until you assign a role to a node group, the group has no defined configuration for the nodes to inherit.

**Organize webserver configurations with roles and profiles**

The **roles and profiles** method is a reliable way to build reusable, configurable, and refactorable system configurations.

Roles and profiles allow you to select relevant pieces of code from modules and bundle them together to create your own custom set of code for managing things. **Profiles** are the individual bundles of code. **Roles** gather profiles together so you can assign them to nodes. This allows you to efficiently organize your Puppet code.

To illustrate roles and profiles, these steps demonstrate how to:

- Define a profile that configures the **example.com** website and includes a firewall rule.

  **Note:** Adding a firewall rule isn’t necessary for an IIS website because the port is already open, but the purpose of this example is to show that you can write a role that manages more than one piece of software (both IIS and the firewall) to accomplish a task.

- Create a role to contain the profile.
- Assign the role to the **iis** node group.

This creates a base structure where, if you had additional websites to serve, you would create additional profiles for those sites. When you have multiple profiles, you can combine profiles within roles or create unique roles for each profile.
Because this example adds a firewall rule, make sure you add the *puppet/windows_firewall* module to your Puppetfile, following the same process you used to Install the *iis* module on page 171. Remember to add the firewall modules dependencies (*puppetlabs/stdlib* and *puppetlabs/registry*), such as:

```
mod 'puppet/windows_firewall', '2.0.2'
mod 'puppetlabs/stdlib', '4.6.0'
mod 'puppetlabs/registry', '1.1.1'
```

**Related information**

The roles and profiles method on page 502

The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.

**Set up your prerequisites**

Before writing content for roles and profiles, you need to create modules to store them in.

1. Create one module for *profile* and one for *role* directly in your control repo. Do not put them in your Puppetfile.

2. Make a new directory in the control repo named *site*. For example, `/etc/puppetlabs/code/environments/production/site`.

3. Add `site` to the modulepath in the `environment.conf` file. The modulepath is the place where Puppet looks for module information. For example: `modulepath = site:modules:$basemodulepath`.

4. Put the *role* and *profile* modules in the *site* directory.

**Write a profile for your IIS website**

Write a webserver profile that includes rules for your *iis_site* and firewall.

**Before you begin**

Make sure you have:

- Installed the *puppetlabs/iis* module, the *puppet/windows_firewall* module, and their dependencies from the Forge.
- Created the *role* and *profile* modules, as explained in Set up your prerequisites.

**Tip:** We recommend writing your code in a code editor, such as VSCode, and then pushing to your Git server. There is a [Puppet VSCode extension](https://vscode.github.io/puppet-vsext/) that supports syntax highlighting of the Puppet language.

1. In the *profile* module, create the following directories and `.pp` file:
   - `manifests/`
   - `webserver/`
   - `example.pp`
2. Paste this Puppet code into the example.pp file:

```puppet
class profile::webserver::example (
  String $content  = 'Hello from iis',
  String $port = '80',
) {

  windows_firewall::exception { 'http':
    ensure       => present,
    direction    => 'in',
    action       => 'allow',
    enabled      => true,
    protocol     => 'TCP',
    local_port   => Integer($port),
    remote_port  => 'any',
    display_name => 'IIS incoming traffic HTTP-In',
    description  => "Inbound rule for IIS web traffic. [TCP ${port}]",
  }

  $iis_features = ['Web-WebServer','Web-Scripting-Tools', 'Web-Mgmt-Console']
  iis_feature { $iis_features:
    ensure => 'present',
  }

  # Delete the default website to prevent a port binding conflict.
  iis_site {'Default Web Site':
    ensure => absent,
    require => Iis_feature['Web-WebServer'],
  }

  iis_site { 'minimal':
    ensure => 'started',
    physicalpath => 'c:\inetpub\minimal',
    applicationpool => 'DefaultAppPool',
    bindings => [
      {
        'bindinginformation' => "${facts['ipaddress']}:${port}:",
        'protocol' => 'TCP',
      },
    ],
    require => [ File['minimal-index'],
      Iis_site['Default Web Site']
    ],
  }

  file { 'minimal':
    ensure => 'directory',
    path => 'c:\inetpub\minimal',
  }

  file { 'minimal-index':
    ensure => 'file',
    path => 'c:\inetpub\minimal\index.html',
    content => $content,
    require => File['minimal']
  }
}
```

This profile applies custom rules for the iis_site class that include settings for $port and $content. The code uses file to ensure the site's main page has content. Finally, there is a firewall rule that only allows traffic from the ports set in the $port setting.

You can add your own code to the profile as needed. For more information, go to these Forge pages:

- IIS module README
- IIS module Reference
Set data for the profile
Hiera is a configuration method that allows you to set defaults in your code or override defaults (in certain circumstances). Use it to refine profile data.

Suppose you want to use the custom fact `stage` to represent the deployment stage of the node, which can be `dev`, `test`, or `prod`. For this example, use `dev` and `prod`.

With Hiera structured data, you can set up a four-layer hierarchy consisting of:

- `console_data` for data defined in the console.
- `nodes/%{trusted.certname}` for per-node overrides.
- `stage/%{facts.stage}` for setting stage-specific data.
- `common` for global fallback data.

This structure lets you tune the settings for ports and IPs in each stage.

For example, to make webservers in the development environment have a custom message and use port 8080, you'd create a data file with the following name, location, and code content:

```yaml
# /etc/puppetlabs/code/environments/production/data/stage/dev.yaml
---
profile::webserver::example::content: "Hello from dev"
profile::webserver::example::ports:
  - '8080'
```

You'd use this code to make webservers in the production environment listen to all interfaces:

```yaml
# /etc/puppetlabs/code/environments/production/data/stage/prod.yaml
---
profile::webserver::example::ips:
  - '0.0.0.0'
  - '::'
```

This is a brief introduction to what you can do with structured data in Hiera. To learn more about setting up hierarchical data, see [Getting started with Hiera](#).

Write a role for your IIS website
Roles contain sets of profiles. To write roles, think about the machines you're managing and decide what else they need in addition to the `webserver` profile.

This example shows how to write a role by combining profiles. In this example, assume you want all nodes in your `iis` node group to use the `webserver` profile you just wrote, and that your organization assigns all machines (including workstations) a profile called `profile::base` that manages basic policies and uses some conditional logic to include operating-system-specific configuration.

1. In your control repo, open the `.pp` file for the `role` module. If it doesn't exist, create the necessary directories and file, such as:

   ```bash
   site-modules\role\manifests\exampleserver.pp
   ```

2. Write a role that includes both the `base` profile and your `webserver` profile:

   ```puppet
   class role::exampleserver {
     include profile::base
     include profile::webserver
   }
   ```

3. You can add more profiles to this role, or create additional roles with more profile configurations based on your needs.
Assign the role to nodes
Assign the `exampleserver` role to the node group containing the nodes that you want to have the `iis_site` configuration you wrote in the `webserver::example` profile.

For this example, assume you want to add `role::exampleserver` to all nodes in the `iis` node group.

1. In the console, click **Node groups** and select the `iis` node group.
2. On the **Classes** tab, select `role::exampleserver` and click **Add class**.
3. Commit the change.

Now, the `iis` node group manages your `iis_site` website based on the rules you wrote in your `webserver` profile. When the nodes check in with PE, PE distributes the role (and the contained profiles) to the individual nodes and ensures the individual nodes have the IIS service and the desired configurations.

Next steps
Now that you have set up some basic automated configuration management with PE, here are some things to do next.

- Check out the [Forge](https://Forge) to download additional modules and start managing other things like NTP or machines running on Azure.
- Learn how to develop high-quality modules with the [Puppet VSCode extension](https://github.com/puppetlabs/puppet-vscode-extension) and [Puppet Development Kit (PDK)](https://puppet.com/pdk).
- See [Configuring Puppet Enterprise](https://docs.puppet.com/puppet-enterprise/5.5/administration.html) on page 279 to fine-tune things like the console performance, orchestration services, Java, and proxy settings.
- Learn more about [Tasks in PE](https://docs.puppet.com/puppet-enterprise/5.5/tasks.html) on page 633 or [Plans in PE](https://docs.puppet.com/puppet-enterprise/5.5/plans.html) on page 663.
- See [Managing access](https://docs.puppet.com/puppet-enterprise/5.5/security-administration.html) on page 331 for information about adding and organizing other PE users and their permissions.
- If you have teams rolling out Puppet code changes across your infrastructure, check out [Getting Started with Continuous Delivery for Puppet Enterprise](https://docs.puppet.com/puppet-enterprise/5.5/continuous-delivery.html).
- Check out our [Youtube channel](https://www.youtube.com/puppet) to learn more about Puppet.

Installing

A Puppet Enterprise deployment typically includes infrastructure components and agents, which are installed on nodes in your environment.

You can install infrastructure components in multiple configurations and scale up with compilers. You can install agents on *nix, Windows, and macOS nodes.
There are several configurations available for Puppet Enterprise. The configuration you use depends on the number of nodes in your environment and the resources required to serve agent catalogs. All PE installations begin with the standard configuration, and scale up by adding additional components as needed.

### Supported architectures

Supported architectures

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Node limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard installation (Recommended)</td>
<td>All infrastructure components are installed on the primary server. This installation type is the easiest to install, upgrade, and troubleshoot.</td>
<td>Up to 2,500</td>
</tr>
<tr>
<td>Large installation</td>
<td>Similar to a standard installation, plus one or more compilers and a load balancer which help distribute the agent catalog compilation workload.</td>
<td>2,500–20,000</td>
</tr>
</tbody>
</table>
## Extra-large installation

The extra-large architecture scales PE deployments to 20,000+ nodes. This architecture is intended to be deployed with the help of Puppet solutions experts. For more information about the capabilities of this architecture and how to deploy it, reach out to your technical account manager.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Node limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra-large installation</td>
<td>Similar to a large installation, plus one or more separate PE-PostgreSQL nodes that run PuppetDB.</td>
<td>20,000+</td>
</tr>
</tbody>
</table>

**Standard installation**

**Large installation**
Standalone PE-PostgreSQL (legacy)

Upgrading from the retired split architecture results in a standalone PE-PostgreSQL architecture. This architecture is similar to a large installation, but with a separate node that hosts the PE-PostgreSQL instance. Standalone PE-PostgreSQL can't be configured with disaster recovery.
System requirements

Refer to these system requirements for Puppet Enterprise installations.

- **Hardware requirements** on page 182
  These hardware requirements are based on internal testing at Puppet and are provided as minimum guidelines to help you determine your hardware needs.
- **Supported operating systems** on page 184
  Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.
- **Supported browsers** on page 190
  The following browsers are supported for use with the console.
- **System configuration** on page 190
  Before installing Puppet Enterprise, make sure that your nodes and network are properly configured.

Hardware requirements

These hardware requirements are based on internal testing at Puppet and are provided as minimum guidelines to help you determine your hardware needs.

Your configuration and code base can significantly affect performance. Use PE tuning and metrics tools to further customize and refine your installation.

### Tip:

If possible, address performance limitations by maximizing your hardware first, then scaling up to the next size architecture as needed. It’s often easier to upgrade your hardware than to add additional infrastructure nodes.

### Related information

**Tuning infrastructure nodes** on page 279
Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

**Puppet Enterprise metrics** on page 414
You can use Puppet Enterprise (PE) metrics for your own performance tuning or provide them to Support for troubleshooting.

**Viewing and managing Puppet Server metrics** on page 415
Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

### Hardware requirements for standard installations

These are the minimum hardware requirements for the primary server in a standard architecture with up to 2,500 nodes.

<table>
<thead>
<tr>
<th>Node volume</th>
<th>Cores</th>
<th>RAM</th>
<th>/opt/</th>
<th>/var/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial use</td>
<td>2</td>
<td>8 GB</td>
<td>20 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>11–100</td>
<td>6</td>
<td>10 GB</td>
<td>50 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>101–500</td>
<td>8</td>
<td>12 GB</td>
<td>50 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>501–1,000</td>
<td>10</td>
<td>16 GB</td>
<td>50 GB</td>
<td>24 GB</td>
</tr>
<tr>
<td>1,000–2,500</td>
<td>12</td>
<td>24 GB</td>
<td>50 GB</td>
<td>24 GB</td>
</tr>
</tbody>
</table>

- **Trial mode**: Although the m5.large instance type is sufficient for trial use, it is not supported. A minimum of four cores is required for production workloads.
- **/opt/ storage requirements**: The database should not exceed 50% of /opt/ to allow for future upgrades.

© 2022 Puppet, Inc., a Perforce company
• **/var/ storage requirements:** There are roughly 20 log files stored in /var/ which are limited in size to 1 GB each. We recommend allocating 24 GB to avoid issues, however log retention settings generally prevent reaching the maximum capacity.

**Hardware requirements for large installations**

These are the minimum hardware requirements for the primary server and compilers in a large architecture with 2,500–20,000 nodes.

Each compiler increases capacity by approximately 1,500–3,000 nodes, until you exhaust the capacity of PuppetDB or the console, which run on the primary server.

<table>
<thead>
<tr>
<th>Node volume</th>
<th>Node</th>
<th>Cores</th>
<th>RAM</th>
<th>/opt/</th>
<th>/var/</th>
<th>EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,500–20,000</td>
<td>Primary node</td>
<td>16</td>
<td>32 GB</td>
<td>150 GB</td>
<td>10 GB</td>
<td>c5.4xlarge</td>
</tr>
<tr>
<td></td>
<td>Each compiler</td>
<td>6</td>
<td>12 GB</td>
<td>30 GB</td>
<td>2 GB</td>
<td>m5.xlarge</td>
</tr>
<tr>
<td></td>
<td>(1,500 - 3,000 nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hardware requirements for extra-large installations**

These are the minimum hardware requirements for the primary server, compilers, and PE-PostgreSQL nodes in an extra-large architecture with 20,000+ nodes.

<table>
<thead>
<tr>
<th>Node volume</th>
<th>Node</th>
<th>Cores</th>
<th>RAM</th>
<th>/opt/</th>
<th>/var/</th>
<th>EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,000+</td>
<td>Primary node</td>
<td>16</td>
<td>32 GB</td>
<td>150 GB</td>
<td>10 GB</td>
<td>c5.4xlarge</td>
</tr>
<tr>
<td></td>
<td>Each compiler</td>
<td>6</td>
<td>12 GB</td>
<td>30 GB</td>
<td>2 GB</td>
<td>m5.xlarge</td>
</tr>
<tr>
<td></td>
<td>(1,500 - 3,000 nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PE-PostgreSQL node</td>
<td>16</td>
<td>128 GB</td>
<td>300 GB</td>
<td>4 GB</td>
<td>r5.4xlarge</td>
</tr>
</tbody>
</table>

If you manage more than 20,000 nodes, contact Puppet professional services to talk about optimizing your setup for your specific requirements.

**Hardware requirements for cloud deployments**

These are the minimum hardware requirements for the primary server in cloud deployments. Cloud deployments use a standard architecture with up to 2,500 nodes.

<table>
<thead>
<tr>
<th>Node volume</th>
<th>AWS</th>
<th>Azure</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial use</td>
<td>m5.large</td>
<td>D2 v4</td>
<td>VM.Standard2.2</td>
</tr>
<tr>
<td>11–100</td>
<td>c5.2xlarge</td>
<td>F8s v2</td>
<td>VM.Standard2.8</td>
</tr>
<tr>
<td>101–500</td>
<td>c5.2xlarge</td>
<td>F8s v2</td>
<td>VM.Standard2.8</td>
</tr>
<tr>
<td>501–1,000</td>
<td>c5.2xlarge</td>
<td>F8s v2</td>
<td>VM.Standard2.8</td>
</tr>
<tr>
<td>1,000–2,500</td>
<td>c5.4xlarge</td>
<td>F16s v2</td>
<td>VM.Standard2.16</td>
</tr>
</tbody>
</table>

Azure and Oracle requirements are not currently tested by Puppet, but are presented here as our best guidance based on comparable AWS testing.
Supported operating systems

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

**Supported operating systems and devices**

You can install PE and the agent on these supported platforms.

For details about platform support lifecycles and planned end-of-life support, see [Platform support lifecycle](#) on the Puppet website.

**Note:** If you use the SELinux security module, you must grant a compliance exception for Puppet and the PXP agent in order for those services to effectively manage configuration.

**Primary server platforms**

The primary server can be installed on the following operating systems. All primary servers require an x86_64 architecture (or amd64 for Ubuntu).

> **CAUTION:** Major primary server OS upgrades (such as Ubuntu 18.04 to 20.04) require [Backing up and restoring PE](#) on page 841.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlmaLinux</td>
<td>8</td>
</tr>
<tr>
<td>CentOS</td>
<td>7</td>
</tr>
<tr>
<td>Oracle Linux</td>
<td>7</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>7, 8</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux (FIPS 140-2 compliant)</td>
<td>7, 8</td>
</tr>
<tr>
<td>Rocky Linux</td>
<td>8</td>
</tr>
<tr>
<td>Scientific Linux</td>
<td>7</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>12, 15</td>
</tr>
<tr>
<td>Ubuntu (General Availability kernels)</td>
<td>18.04, 20.04</td>
</tr>
</tbody>
</table>

**Agent platforms**

The agent can be installed on the following operating systems and architectures.

> **CAUTION:** Major agent OS upgrades (such as Ubuntu 18.04 to 20.04) require reinstalling the `puppet-agent` package (as explained in [Installing agents](#) on page 223) and reinstalling any Ruby plugins/gems that were added at `/opt/puppetlabs/puppet/bin/gem`.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>7.1, 7.2</td>
<td>POWER</td>
</tr>
<tr>
<td>AlmaLinux</td>
<td>8</td>
<td>x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td>Amazon Linux</td>
<td>2</td>
<td>x86_64</td>
</tr>
<tr>
<td>Operating system</td>
<td>Versions</td>
<td>Architecture</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>CentOS</td>
<td>6, 7</td>
<td>• 6: x86_64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 7: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td>Debian</td>
<td>Stretch (9), Buster (10), Bullseye (11)</td>
<td>• Stretch (9): amd64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Buster (10): amd64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bullseye (11): amd64</td>
</tr>
<tr>
<td>Fedora</td>
<td>32, 34</td>
<td>x86_64</td>
</tr>
<tr>
<td>macOS</td>
<td>10.15, 11, 12</td>
<td>• 10.15: x86_64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 11: x86_64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 12: x86_64, M1</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>10, 11</td>
<td>• 10: x86, x64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 11: x64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For FIPS 140-2 compliant Microsoft Windows use version 10 with x64 architecture.</td>
</tr>
<tr>
<td>Microsoft Windows Server</td>
<td>2008 R2, 2012 R2, 2012 R2 Core, 2016, 2016 Core, 2019, 2019 Core, 2022</td>
<td>x64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For FIPS 140-2 compliant Microsoft Windows Server, use 2012 R2 or 2012 R2 Core.</td>
</tr>
<tr>
<td>Oracle Linux</td>
<td>6, 7, 8, 9</td>
<td>• 6: x86_64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 7: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 8: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 9: x86_64</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>6, 7, 8, 9</td>
<td>• 6: x86_64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 7: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 8: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 9: x86_64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For FIPS 140-2 compliant RHEL, use version 7 or 8 with x86_64 architecture.</td>
</tr>
<tr>
<td>Rocky Linux</td>
<td>8</td>
<td>x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td>Operating system</td>
<td>Versions</td>
<td>Architecture</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Scientific Linux</td>
<td>6, 7, 8, 9</td>
<td>• 6: x86_64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 7: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 8: x86_64, aarch64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 9: x86_64</td>
</tr>
<tr>
<td>Solaris</td>
<td>10, 11</td>
<td>SPARC, x86_64</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>11, 12, 15</td>
<td>• 11: x86_64, i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 12: x86_64, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 15: x86_64</td>
</tr>
<tr>
<td>Ubuntu (General Availability kernels)</td>
<td>18.04, 20.04, 22.04</td>
<td>• 18.04: amd64, aarch64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 20.04: amd64, aarch64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 22.04: amd64</td>
</tr>
</tbody>
</table>

**Platform dependencies**

When you install PE or an agent, certain package dependencies are required to ensure the node is operational.

In most cases, dependencies are automatically set up during installation. You might need to manually install dependencies in these cases:

- If you're installing on AIX or Solaris.
- If the node doesn't have internet access.

**Note:** Some operating systems require an active subscription with the vendor's package management system (for example, the Red Hat Network) to install dependencies.

**CentOS dependencies**

<table>
<thead>
<tr>
<th>All nodes</th>
<th>Primary server</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cronie (versions 6 and later)</td>
<td>x</td>
</tr>
<tr>
<td>dmidecode</td>
<td>x</td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
</tr>
<tr>
<td>logrotate</td>
<td>x</td>
</tr>
<tr>
<td>net-tools</td>
<td>x</td>
</tr>
<tr>
<td>pciutils</td>
<td>x</td>
</tr>
<tr>
<td>tar</td>
<td>x</td>
</tr>
<tr>
<td>vixie-cron (version 5)</td>
<td>x</td>
</tr>
<tr>
<td>which</td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td>x</td>
</tr>
<tr>
<td>curl</td>
<td>x</td>
</tr>
<tr>
<td>libjpeg</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>All nodes</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>libtool-ltdl (versions 7 and later)</td>
<td></td>
</tr>
<tr>
<td>libxslt</td>
<td></td>
</tr>
<tr>
<td>mailcap</td>
<td></td>
</tr>
</tbody>
</table>

**RHEL dependencies**

<table>
<thead>
<tr>
<th></th>
<th>All nodes</th>
<th>Primary server</th>
</tr>
</thead>
<tbody>
<tr>
<td>cronie (versions 6 and later)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>dmidecode</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>logrotate</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>net-tools</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>pciutils</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>tar</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>vixie-cron (version 5)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>which</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>curl</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libjpeg</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libtool-ltdl (versions 7 and later)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libxslt</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>mailcap</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

**SUSE Linux Enterprise Server dependencies**

**Tip:** If you encounter problems installing dependencies, inspect the error messages for packages that require other SUSE Linux Enterprise Server packaging modules to be enabled, and use `zypper package-search <PACKAGE_NAME>` to locate them for manual installation.

<table>
<thead>
<tr>
<th></th>
<th>All nodes</th>
<th>Primary server</th>
</tr>
</thead>
<tbody>
<tr>
<td>cron</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libxslt</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>logrotate</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>net-tools</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>pciutils</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>pmtools</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>tar</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>curl</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
### All nodes

<table>
<thead>
<tr>
<th>db43</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>libjpeg</td>
<td>x</td>
</tr>
<tr>
<td>unixODBC</td>
<td>x</td>
</tr>
</tbody>
</table>

### Primary server

#### Ubuntu dependencies

<table>
<thead>
<tr>
<th>cron</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmkdir</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>gnupg</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>hostname</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libldap-2.4-2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libreadline5</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>logrotate</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>pciutils</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>tar</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ca-certificates-java</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>curl</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libcap2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libgtk2.0-0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libjpeg62</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libmagic1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>liboss-server-uuid16</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libper3</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libxslt1.1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>mime-support</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>perl</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

#### AIX dependencies and limitations
Before installing the agent on AIX systems, install these packages.

- bash
- zlib
- readline
- curl
- openssl
- tar

© 2022 Puppet, Inc., a Perforce company
CAUTION: For curl and openssl packages, you must use the versions provided by the "AIX Toolbox Cryptographic Content" repository, which is available via IBM support. Note that the cURL version must be 7.9.3. Do not use the cURL version in the AIX toolbox package for Linux applications, as that version does not include support for OpenSSL.

To install the bash, zlib, and readline packages on a node directly, run `rpm -Uvh` with the following URLs. The RPM package provider must be run as root.

- ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/readline/readline-4.3-2.aix5.1.ppc.rpm (AIX 5.3 only)

If you are behind a firewall or running an http proxy, the above commands might not work. Instead, use the AIX toolbox packages download available from IBM.

GPG verification does not work on AIX, because the RPM version it uses is too old. The AIX package provider doesn't support package downgrades (installing an older package over a newer package). Avoid using leading zeros when specifying a version number for the AIX provide, for example, use 2.3.4 not 02.03.04.

The PE AIX implementation supports the NIM, BFF, and RPM package providers. Check the type reference for technical details on these providers.

**Solaris dependencies and limitations**

Solaris support is agent only.

For Solaris 10, these packages are required:

- tar
- SUNWgccruntime
- SUNWzlib
- In some instances, bash might not be present on Solaris systems. It needs to be installed before running the installer. Install bash via the media used to install the operating system, or via CSW if that is present on your system. (CSWbash or SUNWbash are both suitable.

For Solaris 11 these packages are required:

- tar
- readline
- openssl

These packages are available in the Solaris release repository, which is enabled by default in version 11. The installer automatically installs these packages; however, if the release repository is not enabled, the packages must be installed manually.

**Upgrade your operating system with PE installed**

If you have PE installed, take extra precautions before performing a major upgrade of your machine's operating system.

Performing major upgrades of your operating system with PE installed can cause errors and issues with PE. A major operating system upgrade is an upgrade to a new whole version, such as an upgrade from RHEL 6.0 to 7.0; it does not refer to a minor version upgrade, like RHEL 6.5 to 6.6. Major upgrades typically require a new version of PE.

1. Back up your databases and other PE files.
2. Perform a complete uninstall (using the `-p` and `-d` uninstaller options).
3. Upgrade your operating system.
4. Install PE.
5. Restore your backup.
**Related information**

**Back up your infrastructure** on page 842
PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

**Restore your infrastructure** on page 842
Use the restore commands to migrate your primary server to a new host or to recover from system failure.

**Uninstalling** on page 254
Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

**Installing PE** on page 209
Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

**Supported browsers**
The following browsers are supported for use with the console.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome</td>
<td>Current version as of release</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>Current version as of release</td>
</tr>
<tr>
<td>Microsoft Edge</td>
<td>Current version as of release</td>
</tr>
<tr>
<td>Apple Safari</td>
<td>10 or later</td>
</tr>
</tbody>
</table>

**System configuration**
Before installing Puppet Enterprise, make sure that your nodes and network are properly configured.

**Note:** Port numbers are Transmission Control Protocols (TCP), unless noted otherwise.

**Network considerations**
Before installing, consider these network requirements

**Timekeeping**
Use NTP or an equivalent service to ensure that time is in sync between your primary server, which acts as the certificate authority, and any agent nodes. If time drifts out of sync in your infrastructure, you might encounter issues such as agents receiving outdated certificates. A service like NTP (available as a supported module) ensures accurate timekeeping.

**Name resolution**
Decide on a preferred name or set of names that agent nodes can use to contact the primary server. Ensure that the primary server can be reached by domain name lookup by all future agent nodes.

You can simplify configuration of agent nodes by using a CNAME record to make the primary server reachable at the hostname `puppet`, which is the default primary server hostname that is suggested when installing an agent node.

**Web URLs used for deployment and management**
PE uses some external web URLs for certain deployment and management tasks. You might want to ensure these URLs are reachable from your network prior to installation, and be aware that they might be called at various stages of configuration.
Antivirus and antimalware considerations
Antivirus and antimalware software can impact or prevent the proper functioning of PE. While we don’t have an official stance on which antivirus and antimalware product you use, here are some considerations to keep in mind while installing and configuring antivirus and antimalware software.

- Exclude the /etc/puppetlabs and /opt/puppetlabs directories from antivirus and antimalware tools that scan disk write operations to avoid performance issues.
- Some antivirus and antimalware software requires a lot of system processing power. Tune your system resources to accommodate the software so it doesn’t slow your performance.
- Some antivirus and antimalware software defaults to using port 8081, which is the same port PuppetDB uses. When installing the software, consider which port it uses so it doesn’t conflict with PuppetDB communications.
- For agents, you can exclude C:\ProgramData\PuppetLabs\pe_patch if your antivirus is holding a lock on log files and causing patching failures.

Related information
Tuning infrastructure nodes on page 279
Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

The default port for PuppetDB conflicts with another service on page 852
By default, PuppetDB communicates over port 8081. In some cases, this might conflict with existing services, for example McAfee ePolicy Orchestrator.

**Firewall configuration**
Follow these guidelines for firewall configuration based on your installation type.

**Firewall configuration for standard installations**
These are the port requirements for standard installations.

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
<th>VPC access in cloud deployments</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>- Code Manager uses this port to tell a git to clone and fetch content via SSH.</td>
<td>External</td>
</tr>
<tr>
<td>Port</td>
<td>Use</td>
<td>VPC access in cloud deployments</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>---------------------------------</td>
</tr>
</tbody>
</table>
| 443  | • Code Manager uses this port to tell a git to clone and fetch content via HTTPS.  
     • This port provides host access to the console  
     • The console accepts HTTPS traffic from end users on this port.  
     • Classifier group: **PE Console** | **External** |
| 4433 | • This port is used as a classifier / console services API endpoint.  
     • The primary server communicates with the console over this port.  
     • Classifier group: **PE Console** | **External** |
| 5432 | • This port is used to replicate PostgreSQL data between the primary server and replica. | **Internal** |
| 8081 | • PuppetDB accepts traffic/requests on this port.  
     • The primary server and console send traffic to PuppetDB on this port.  
     • PuppetDB status checks are sent over this port.  
     • Classifier group: **PE PuppetDB** | **Internal** |
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
<th>VPC access in cloud deployments</th>
</tr>
</thead>
</table>
| 8140 | • The primary server uses this port to accept inbound traffic/requests from agents.  
• The console sends requests to the primary server on this port.  
• Certificate requests are passed over this port unless `ca_port` is set differently.  
• Puppet Server status checks are sent over this port.  
• Classifier group: **PE Master** | Internal |
| 8142 | • Orchestrator and the **Run Puppet** button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent.  
• Classifier group: **PE Orchestrator** | Internal |
| 8143 | • Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation.  
• Classifier group: **PE Orchestrator** | Internal |
| 8170 | • Code Manager uses this port to deploy environments, run webhooks, and make API calls. | Internal |

**Firewall configuration for large installations**

These are the port requirements for large installations with compilers.
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>• Code Manager uses this port to tell a git to clone and fetch content via SSH.</td>
</tr>
</tbody>
</table>
| 443  | • Code Manager uses this port to tell a git to clone and fetch content via HTTPS.  
• This port provides host access to the console  
• The console accepts HTTPS traffic from end users on this port.  
• Classifier group: **PE Console** |
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 4433 | • This port is used as a classifier / console services API endpoint.  
• The primary server communicates with the console over this port.  
• Classifier group: **PE Console** |
| 5432 | • This port is used to replicate PostgreSQL data between the primary server and replica.  
• The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL. |
| 8081 | • PuppetDB accepts traffic/requests on this port.  
• The primary server and console send traffic to PuppetDB on this port.  
• PuppetDB status checks are sent over this port.  
• Classifier group: **PE PuppetDB** |
| 8140 | • The primary server uses this port to accept inbound traffic/requests from agents.  
• The console sends requests to the primary server on this port.  
• Certificate requests are passed over this port unless `ca_port` is set differently.  
• Puppet Server status checks are sent over this port.  
• The primary server uses this port to send status checks to compilers. (Not required to run PE.)  
• Classifier group: **PE Master** |
| 8142 | • Orchestrator and the **Run Puppet** button use this port on the primary server to accept inbound traffic/ responses from agents via the Puppet Execution Protocol agent.  
• Classifier group: **PE Orchestrator** |
## Installing 197

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8143 | • Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation.   
• Classifier group: **PE Orchestrator** |
| 8170 | • Code Manager uses this port to deploy environments, run webhooks, and make API calls. |

**Firewall configuration for extra-large installations**

These are the port requirements for extra-large installations with compilers.
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>• Code Manager uses this port to tell a git to clone and fetch content via SSH.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 443  | • Code Manager uses this port to tell a git to clone and fetch content via HTTPS.  
      • This port provides host access to the console  
      • The console accepts HTTPS traffic from end users on this port.  
      • Classifier group: **PE Console** |
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 4433  | • This port is used as a classifier / console services API endpoint.  
      • The primary server communicates with the console over this port.  
      • Classifier group: **PE Console**  |
| 5432  | • The primary server and replica use this port to replicate PostgreSQL data on PE-PostgreSQL nodes.  
      • The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL.  |
| 8081  | • PuppetDB accepts traffic/requests on this port.  
      • The primary server and console send traffic to PuppetDB on this port.  
      • PuppetDB status checks are sent over this port.  
      • Classifier group: **PE PuppetDB**  |
| 8140  | • The primary server uses this port to accept inbound traffic/requests from agents.  
      • The console sends requests to the primary server on this port.  
      • Certificate requests are passed over this port unless `ca_port` is set differently.  
      • Puppet Server status checks are sent over this port.  
      • The primary server uses this port to send status checks to compilers. (Not required to run PE.)  
      • Classifier group: **PE Master**  |
| 8142  | • Orchestrator and the **Run Puppet** button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent.  
      • Classifier group: **PE Orchestrator**  |
### Firewall configuration for standalone PE-PostgreSQL installations

These are the port requirements for installations with compilers and standalone PE-PostgreSQL installations.

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8143  | • Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation.  
  • Classifier group: **PE Orchestrator**   |
| 8170  | • Code Manager uses this port to deploy environments, run webhooks, and make API calls.   |
| 22    | • Code Manager uses this port to tell a git to clone and fetch content via SSH.   |

© 2022 Puppet, Inc., a Perforce company
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 443  | - Code Manager uses this port to tell a git to clone and fetch content via HTTPS.  
- This port provides host access to the console  
- The console accepts HTTPS traffic from end users on this port.  
- Classifier group: **PE Console** |
| 4433 | - This port is used as a classifier / console services API endpoint.  
- The primary server communicates with the console over this port.  
- Classifier group: **PE Console** |
| 5432 | - The standalone PE-PostgreSQL node uses this port to accept inbound traffic/requests from the primary server.  
- The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL. |
| 8081 | - PuppetDB accepts traffic/requests on this port.  
- The primary server and console send traffic to PuppetDB on this port.  
- PuppetDB status checks are sent over this port.  
- Classifier group: **PE PuppetDB** |
| 8140 | - The primary server uses this port to accept inbound traffic/requests from agents.  
- The console sends requests to the primary server on this port.  
- Certificate requests are passed over this port unless `ca_port` is set differently.  
- Puppet Server status checks are sent over this port.  
- The primary server uses this port to send status checks to compilers. (Not required to run PE.)  
- Classifier group: **PE Master** |
### What gets installed and where?

Puppet Enterprise installs several software components, configuration files, databases, services and users, and log files. It's useful to know the locations of these should you ever need to troubleshoot or manage your infrastructure.

#### Software components installed

PE installs several software components and dependencies. These tables show which version of each component is installed for 2019.y and 2018.y releases.

The functional components of the software are separated between those packaged with the agent and those packaged on the server side (which also includes the agent).

**Note:** PE also installs other dependencies, as documented in the system requirements.

This table shows the components installed on all agent nodes.

**Note:** Hiera 5 is a backwards-compatible evolution of Hiera, which is built into Puppet 4.9.0 and higher. To provide some backwards-compatible features, it uses the classic Hiera 3.x.x codebase version listed in this table.

<table>
<thead>
<tr>
<th>PE Version</th>
<th>Puppet Agent</th>
<th>Puppet</th>
<th>Facter</th>
<th>Hiera</th>
<th>Resource API</th>
<th>MCollectiveRuby</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.12</td>
<td>6.28.0</td>
<td>6.28.0</td>
<td>3.14.24</td>
<td>3.9.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>(overlap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>support)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019.8.11</td>
<td>6.27.1</td>
<td>6.27.0</td>
<td>3.14.23</td>
<td>3.9.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.10</td>
<td>6.26.0</td>
<td>6.26.0</td>
<td>3.14.22</td>
<td>3.8.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.9</td>
<td>6.25.1</td>
<td>6.25.1</td>
<td>3.14.21</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.8</td>
<td>6.24.0</td>
<td>6.24.0</td>
<td>3.14.19</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.7</td>
<td>6.23.0</td>
<td>6.23.0</td>
<td>3.14.18</td>
<td>3.7.0</td>
<td>1.8.14</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
<tr>
<td>2019.8.6</td>
<td>6.22.1</td>
<td>6.22.1</td>
<td>3.14.17</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.9</td>
</tr>
</tbody>
</table>

---

© 2022 Puppet, Inc., a Perforce company
<table>
<thead>
<tr>
<th>PE Version</th>
<th>Puppet Agent</th>
<th>Puppet</th>
<th>Facter</th>
<th>Hiera</th>
<th>Resource API</th>
<th>MCollective</th>
<th>Ruby</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.5</td>
<td>6.21.1</td>
<td>6.21.1</td>
<td>3.14.16</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1i</td>
</tr>
<tr>
<td>2019.8.4</td>
<td>6.19.1</td>
<td>6.19.1</td>
<td>3.14.14</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2019.8.3</td>
<td>6.19.1</td>
<td>6.19.1</td>
<td>3.14.14</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2019.8.1</td>
<td>6.17.0</td>
<td>6.17.0</td>
<td>3.14.12</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2019.8</td>
<td>6.16.0</td>
<td>6.16.0</td>
<td>3.14.11</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2019.7</td>
<td>6.15.0</td>
<td>6.15.0</td>
<td>3.14.10</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.8</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2019.5</td>
<td>6.14.0</td>
<td>6.14.0</td>
<td>3.14.9</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.7</td>
<td>1.1.1d</td>
</tr>
<tr>
<td>2019.4</td>
<td>6.14.0</td>
<td>6.14.0</td>
<td>3.14.9</td>
<td>3.6.0</td>
<td>1.8.13</td>
<td>N/A</td>
<td>2.5.7</td>
<td>1.1.1d</td>
</tr>
<tr>
<td>2019.3</td>
<td>6.12.0</td>
<td>6.12.0</td>
<td>3.14.7</td>
<td>3.6.0</td>
<td>1.8.11</td>
<td>N/A</td>
<td>2.5.7</td>
<td>1.1.1d</td>
</tr>
<tr>
<td>2018.1.18</td>
<td>5.5.22</td>
<td>5.5.22</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2018.1.17</td>
<td>5.5.22</td>
<td>5.5.22</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2018.1.16</td>
<td>5.5.21</td>
<td>5.5.21</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.5.20</td>
<td>5.5.20</td>
<td>3.11.13</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.0.2u</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.5.19</td>
<td>5.5.19</td>
<td>3.11.12</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2u</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.5.18</td>
<td>5.5.18</td>
<td>3.11.11</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2t</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.5.17</td>
<td>5.5.17</td>
<td>3.11.10</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2t</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.5.16</td>
<td>5.5.16</td>
<td>3.11.9</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2r</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.5.14</td>
<td>5.5.14</td>
<td>3.11.8</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2r</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.5.10</td>
<td>5.5.10</td>
<td>3.11.7</td>
<td>3.4.6</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.5.8</td>
<td>5.5.8</td>
<td>3.11.6</td>
<td>3.4.5</td>
<td>N/A</td>
<td>2.12.4</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.1.4</td>
<td>5.5.6</td>
<td>5.5.6</td>
<td>3.11.4</td>
<td>3.4.4</td>
<td>N/A</td>
<td>2.12.3</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.1.3</td>
<td>5.5.4</td>
<td>5.5.3</td>
<td>3.11.3</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.2</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.1.2</td>
<td>5.5.3</td>
<td>5.5.2</td>
<td>3.11.2</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.2</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.1.0</td>
<td>5.5.1</td>
<td>5.5.1</td>
<td>3.11.1</td>
<td>3.4.3</td>
<td>N/A</td>
<td>2.12.1</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
</tbody>
</table>

This table shows components installed on server nodes.

<table>
<thead>
<tr>
<th>PE Version</th>
<th>Puppet Server</th>
<th>PuppetDB</th>
<th>r10k</th>
<th>Razor</th>
<th>Bolt Services</th>
<th>Agentless</th>
<th>PostgreSQL</th>
<th>Java</th>
<th>ActiveMQ</th>
<th>Nginx</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.12</td>
<td>6.20.0</td>
<td>6.22.1</td>
<td>3.15.1</td>
<td>N/A</td>
<td>3.25.0</td>
<td>1.2.4</td>
<td>11.17</td>
<td>11.0</td>
<td>N/A</td>
<td>1.22.0</td>
</tr>
<tr>
<td>(overlap support)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019.8.11</td>
<td>6.19.0</td>
<td>6.21.0</td>
<td>3.14.0</td>
<td>N/A</td>
<td>3.22.1</td>
<td>1.2.4</td>
<td>11.15</td>
<td>11.0</td>
<td>N/A</td>
<td>1.21.0</td>
</tr>
<tr>
<td>2019.8.10</td>
<td>6.18.0</td>
<td>6.20.2</td>
<td>3.14.0</td>
<td>N/A</td>
<td>3.21.0</td>
<td>1.2.4</td>
<td>11.13</td>
<td>11.0</td>
<td>N/A</td>
<td>1.21.0</td>
</tr>
<tr>
<td>2019.8.9</td>
<td>6.17.1</td>
<td>6.19.1</td>
<td>3.13.0</td>
<td>N/A</td>
<td>3.20.0</td>
<td>1.2.4</td>
<td>11.13</td>
<td>11.0</td>
<td>N/A</td>
<td>1.21.0</td>
</tr>
<tr>
<td>2019.8.8</td>
<td>6.16.1</td>
<td>6.18.2</td>
<td>3.10.0</td>
<td>N/A</td>
<td>3.13.0</td>
<td>1.2.4</td>
<td>11.13</td>
<td>11.0</td>
<td>N/A</td>
<td>1.21.0</td>
</tr>
</tbody>
</table>
| PE Version | Puppet Server | PuppetDB r10k | Razor Server | Bolt Services | Agentless PostgreSQL Executor (ACE) Services | ActiveMQ
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.7</td>
<td>6.16.0</td>
<td>6.17.0</td>
<td>3.9.2</td>
<td>N/A</td>
<td>3.10.0</td>
<td>1.2.4</td>
</tr>
<tr>
<td>2019.8.6</td>
<td>6.15.3</td>
<td>6.16.1</td>
<td>3.9.0</td>
<td>N/A</td>
<td>3.7.1</td>
<td>1.2.4</td>
</tr>
<tr>
<td>2019.8.5</td>
<td>6.15.1</td>
<td>6.15.0</td>
<td>3.8.0</td>
<td>N/A</td>
<td>3.0.0</td>
<td>1.2.2</td>
</tr>
<tr>
<td>2019.8.4</td>
<td>6.14.1</td>
<td>6.13.1</td>
<td>3.6.0</td>
<td>N/A</td>
<td>2.32.0</td>
<td>1.2.1</td>
</tr>
<tr>
<td>2019.8.3</td>
<td>6.14.1</td>
<td>6.13.1</td>
<td>3.6.0</td>
<td>N/A</td>
<td>2.32.0</td>
<td>1.2.1</td>
</tr>
<tr>
<td>2019.8.1</td>
<td>6.12.1</td>
<td>6.11.3</td>
<td>3.5.2</td>
<td>N/A</td>
<td>2.16.0</td>
<td>1.2.0</td>
</tr>
<tr>
<td>2019.8</td>
<td>6.12.0</td>
<td>6.11.1</td>
<td>3.5.1</td>
<td>N/A</td>
<td>2.11.1</td>
<td>1.2.0</td>
</tr>
<tr>
<td>2019.7</td>
<td>6.11.1</td>
<td>6.10.1</td>
<td>3.5.0</td>
<td>1.9.7</td>
<td>2.6.0</td>
<td>1.1.1</td>
</tr>
<tr>
<td>2019.5</td>
<td>6.9.2</td>
<td>6.9.1</td>
<td>3.4.1</td>
<td>1.9.7</td>
<td>1.49.0</td>
<td>1.1.1</td>
</tr>
<tr>
<td>2019.4</td>
<td>6.9.1</td>
<td>6.9.1</td>
<td>3.4.1</td>
<td>1.9.7</td>
<td>1.49.0</td>
<td>1.1.1</td>
</tr>
<tr>
<td>2019.3</td>
<td>6.8.0</td>
<td>6.8.1</td>
<td>3.4.0</td>
<td>1.9.6</td>
<td>1.45.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018.1.18</td>
<td>5.3.16</td>
<td>5.2.19</td>
<td>2.6.9</td>
<td>1.9.9</td>
<td>N/A</td>
<td>9.6.20</td>
</tr>
<tr>
<td>2018.1.17</td>
<td>5.3.16</td>
<td>5.2.19</td>
<td>2.6.9</td>
<td>1.9.9</td>
<td>N/A</td>
<td>9.6.19</td>
</tr>
<tr>
<td>2018.1.16</td>
<td>5.3.14</td>
<td>5.2.18</td>
<td>2.6.8</td>
<td>1.9.9</td>
<td>N/A</td>
<td>9.6.18</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.3.13</td>
<td>5.2.15</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>N/A</td>
<td>9.6.17</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.3.12</td>
<td>5.2.13</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>N/A</td>
<td>9.6.17</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.3.11</td>
<td>5.2.12</td>
<td>2.6.7</td>
<td>1.9.6</td>
<td>N/A</td>
<td>9.6.16</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.3.10</td>
<td>5.2.11</td>
<td>2.6.7</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.15</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.3.9</td>
<td>5.2.9</td>
<td>2.6.6</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.13</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.3.8</td>
<td>5.2.8</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.12</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.3.7</td>
<td>5.2.7</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.3.6</td>
<td>5.2.6</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.4</td>
<td>5.3.5</td>
<td>5.2.4</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.10</td>
</tr>
<tr>
<td>2018.1.3</td>
<td>5.3.4</td>
<td>5.2.4</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.8</td>
</tr>
<tr>
<td>2018.1.2</td>
<td>5.3.3</td>
<td>5.2.2</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>N/A</td>
<td>9.6.8</td>
</tr>
<tr>
<td>2018.1.0</td>
<td>5.3.2</td>
<td>5.2.2</td>
<td>2.6.2</td>
<td>1.8.1</td>
<td>N/A</td>
<td>9.6.8</td>
</tr>
</tbody>
</table>

**Executable binaries and symlinks installed**

PE installs executable binaries and symlinks for interacting with tools and services.

On *nix nodes, all software is installed under `/opt/puppetlabs`.

On Windows nodes, all software is installed in Program Files at Puppet Labs\Puppet.

Executable binaries on *nix are in `/opt/puppetlabs/bin` and `/opt/puppetlabs/sbin`. 
**Tip:** To include binaries in your default $PATH, manually add them to your profile or export the path:

```bash
export PATH=$PATH:/opt/puppetlabs/bin
```

To make essential Puppet tools available to all users, the installer automatically creates symlinks in `/usr/local/bin` for the `facter`, `puppet`, `pe-man`, `r10k`, and `hiera` binaries. Symlinks are created only if `/usr/local/bin` is writeable. Users of AIX and Solaris versions 10 and 11 must add `/usr/local/bin` to their default path.

For macOS agents, symlinks aren’t created until the first successful run that applies the agents’ catalogs.

**Tip:** You can disable symlinks by changing the `manage_symlinks` setting in your default Hiera file:

```yaml
puppet_enterprise::manage_symlinks: false
```

Binaries provided by other software components, such as those for interacting with the PostgreSQL server, PuppetDB, or Ruby packages, do not have symlinks created.

---

**Modules and plugins installed**

PE installs modules and plugins for normal operations.

Modules included with the software are installed on the primary server in `/opt/puppetlabs/puppet/modules`. Don’t modify anything in this directory or add modules of your own. Instead, install non-default modules in `/etc/puppetlabs/code/environments/<environment>/modules`.

---

**Configuration files installed**

PE installs configuration files that you might need to interact with from time to time.

On *nix nodes, configuration files live at `/etc/puppetlabs/puppet`.

On Windows nodes, configuration files live at `<COMMON_APPDATA>\PuppetLabs`. The location of this folder varies by Windows version; in 2008 and 2012, its default location is `C:\ProgramData\PuppetLabs\puppet\etc`.

The software's `confdir` is in the `puppet` subdirectory. This directory contains the `puppet.conf` file, `auth.conf`, and the SSL directory.

---

**Tools installed**

PE installs several suites of tools to help you work with the major components of the software.

- **Puppet tools** — Tools that control basic functions of the software such as `puppet master` and `puppet cert`.
- **Client tools** — The `pe-client-tools` package collects a set of CLI tools that extend the ability for you to access services from the primary server or a workstation. This package includes:
  - **Orchestrator** — The orchestrator is a set of interactive command line tools that provide the interface to the orchestration service. Orchestrator also enables you to enforce change on the environment level. Tools include `puppet job` and `puppet task`.
  - **Puppet Access** — Users can generate tokens to authenticate their access to certain command line tools and API endpoints.
  - **Code Manager CLI** — The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.
  - **PuppetDB CLI** — This a tool for working with PuppetDB, including building queries and handling exports.
- **Module tool** — The module tool is used to access and create modules, which are reusable chunks of Puppet code users have written to automate configuration and deployment tasks. For more information, and to access modules, visit the Forge.

---

© 2022 Puppet, Inc., a Perforce company
• **Console** — The console is the web user interface for PE. The console provides tools to view and edit resources on your nodes, view reports and activity graphs, and more.

**Databases installed**

PE installs several default databases, all of which use PostgreSQL as a database backend.

The PE PostgreSQL database includes the following databases:

<table>
<thead>
<tr>
<th>Database</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-activity</td>
<td>Activity data from the classifier, including who, what, and when</td>
</tr>
<tr>
<td>pe-classifier</td>
<td>Classification data, all node group information</td>
</tr>
<tr>
<td>pe-inventory</td>
<td>Connection information and credentials for agentless node connections</td>
</tr>
<tr>
<td>pe-orchestrator</td>
<td>Orchestrator data, including details about job runs</td>
</tr>
<tr>
<td>pe-puppetdb</td>
<td>PuppetDB data, including exported resources, catalogs, facts, and reports</td>
</tr>
<tr>
<td>pe-rbac</td>
<td>RBAC data, including users, permissions, and AD/LDAP info</td>
</tr>
</tbody>
</table>

Use the native PostgreSQL tools to perform database exports and imports. At a minimum, perform backups to a remote system nightly, or as dictated by your company policy.

**Services installed**

PE installs several services used to interact with the software during normal operations.

<table>
<thead>
<tr>
<th>Service</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-console-services</td>
<td>Manages and serves the console.</td>
</tr>
<tr>
<td>pe-puppetserver</td>
<td>Runs the primary server.</td>
</tr>
<tr>
<td>pe-nginx</td>
<td>Nginx, serves as a reverse-proxy to the console.</td>
</tr>
<tr>
<td>puppet</td>
<td>(on Enterprise Linux and Debian-based platforms) Runs the agent daemon on every agent node.</td>
</tr>
<tr>
<td>pe-puppetdb, pe-postgresql</td>
<td>Daemons that manage and serve the database components. The pe-postgresql service is created only if the software installs and manages PostgreSQL.</td>
</tr>
<tr>
<td>pxp-agent</td>
<td>Runs the Puppet Execution Protocol agent process.</td>
</tr>
<tr>
<td>pe-orchestration-services</td>
<td>Runs the orchestration process.</td>
</tr>
<tr>
<td>pe-ace-server</td>
<td>Runs the ace server.</td>
</tr>
<tr>
<td>pe-bolt-server</td>
<td>Runs the Bolt server.</td>
</tr>
</tbody>
</table>

**User and group accounts installed**

These are the user and group accounts installed.

<table>
<thead>
<tr>
<th>User</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-puppet</td>
<td>Runs the primary server processes spawned by pe-puppetserver.</td>
</tr>
<tr>
<td>pe-webserver</td>
<td>Runs Nginx.</td>
</tr>
<tr>
<td>User</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pe-puppetdb</td>
<td>Has root access to the database.</td>
</tr>
<tr>
<td>pe-postgres</td>
<td>Has access to the pe-postgreSQL instance. Created only if the software installs and manages PostgreSQL.</td>
</tr>
<tr>
<td>pe-console-services</td>
<td>Runs the console process.</td>
</tr>
<tr>
<td>pe-orchestration-services</td>
<td>Runs the orchestration process.</td>
</tr>
<tr>
<td>pe-ace-server</td>
<td>Runs the ace server.</td>
</tr>
<tr>
<td>pe-bolt-server</td>
<td>Runs the Bolt server.</td>
</tr>
</tbody>
</table>

**Log files installed**

The software distributed with PE generates log files that you can collect for compliance or use for troubleshooting.

**Primary server logs**

The primary server has these logs.

- `/var/log/puppetlabs/puppetserver/code-manager-access.log`
- `/var/log/puppetlabs/puppetserver/file-sync-access.log`
- `/var/log/puppetlabs/puppetserver/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on compilers.
- `/var/log/puppetlabs/puppetserver/puppetserver.log` — The primary server logs its activity, including compilation errors and deprecation warnings, here.
- `/var/log/puppetlabs/puppetserver/puppetserver-access.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/puppetserver/puppetserver-status.log`

**Agent logs**

The locations of agent logs depend on the agent operating system.

On *nix nodes, the agent service logs its activity to the syslog service. Your syslog configuration dictates where these messages are saved, but the default location is `/var/log/messages` on Linux, `/var/log/system.log` on macOS, and `/var/adm/messages` on Solaris.

On Windows nodes, the agent service logs its activity to the event log. You can view its logs by browsing the event viewer.

**Console and console services logs**

The console and pe-console-services has these logs.

- `/var/log/puppetlabs/console-services/console-services.log`
- `/var/log/puppetlabs/console-services/console-services-api-access.log`
- `/var/log/puppetlabs/console-services-console-services-access.log`
- `/var/log/puppetlabs/console-services-console-services-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/nginx/access.log`
- `/var/log/puppetlabs/nginx/error.log` — Contains errors related to nginx. Console errors that aren’t logged elsewhere can be found in this log.

**Installer logs**

The installer has these logs.
Installing

- `/var/log/puppetlabs/installer/http.log` — Contains web requests sent to the installer. This log is present only on the machine from which a web-based installation was performed.
- `/var/log/puppetlabs/installer/orchestrator_info.log` — Contains run details about puppet infrastructure commands that use the orchestrator, including the commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.
- `/var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log` — Contains the contents of the last installer run.
- `/var/log/puppetlabs/installer/installer-<timestamp>.log` — Contains the operations performed and any errors that occurred during installation.
- `/var/log/puppetlabs/installer/<action_name>_<timestamp>_<run_description>.log` — Contains details about disaster recovery command execution. Each action triggers multiple Puppet runs, some on the primary server, some on the replica, so there might be multiple log files for each command.

**Database logs**

The database has these logs.

- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/pgstartup.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Mon.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Tue.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Wed.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Thu.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Fri.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Sat.log`
- `/var/log/puppetlabs/postgresql/<POSTGRESQL VERSION NUMBER>/postgresql-Sun.log`
- `/var/log/puppetlabs/puppetdb/puppetdb.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-access.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-status.log`

**Orchestration logs**

The orchestration service and related components have these logs.

- `/var/log/puppetlabs/orchestration-services/aggregate-node-count.log`
- `/var/log/puppetlabs/orchestration-services/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on the primary server.
- `/var/log/puppetlabs/orchestration-services/pcp-broker-access.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services-access.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/orchestration-services/orchestration-services-daemon.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services-status.log`
- `/var/log/puppetlabs/pxp-agent/pxp-agent.log` (on *nix) or `C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log` (on Windows) — Contains the Puppet Execution Protocol agent log file.

© 2022 Puppet, Inc., a Perforce company
Certificates installed

During installation, the software generates and installs a number of SSL certificates so that agents and services can authenticate themselves.

These certs can be found at `/etc/puppetlabs/puppet/ssl/certs`.

A certificate with the same name as the agent that runs on the primary server is generated during installation. This certificate is used by PuppetDB and the console.

Services that run on the primary server — for example, `pe-orchestration-services` and `pe-console-services` — use the agent certificate to authenticate.

Secret key file installed

During installation, the software generates a secret key file that is used to encrypt and decrypt sensitive data stored in the inventory service.

The secret key is stored at: `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`

Installing PE

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

To install a FIPS-enabled PE primary server, install the appropriate FIPS-enabled PE tarball, for example `puppet-enterprise-2019.8.12-redhatfips-7-x86_64.tar`, on a third party supported platform with FIPS mode enabled. The node must be configured with sufficient available entropy for the installation process to succeed.

Verify the installation package

These steps are only required if your organization requires you to verify authenticity before installing packages. These steps explain how to use GnuPG (GPG) to verify the PE installation tarball.

**Before you begin**

You must have GnuPG (GPG) installed to be able to sign for the release key. GPG is an open source program that allows you to safely encrypt and sign digital communications. You can download GPG from the GnuPG website or use your package management system to install it by running something like: `yum install gnupg`

1. Download the tarball appropriate to your operating system and architecture.

   **Tip:** To download packages from the command line, run `wget --content-disposition <URL>` or `curl -JLO <URL>`, using the URL for the tarball you want to download.

2. To import the Puppet public key, run:

   ```
   uri='https://downloads.puppet.com/puppet-gpg-signing-key-20250406.pub'
   curl "$uri" | gpg --import
   ```

   See Usage notes for curl examples for information about forming curl commands.
3. To print the key fingerprint, run:

   ```
gpg --fingerprint 0x4528B6CD9E61EF26
   
   This command returns the primary key fingerprint, such as:
   ```

   D681 1ED3 ADEE B844 1AF5 AA8F 4528 B6CD 9E61 EF26

4. Download the **GPG SIGNATURE .asc** file corresponding to your PE tarball. You can find links to these files on the PE Download page.

5. To verify the installation package release signature, run:

   ```
gpg --verify puppet-enterprise-<version>-<platform>.tar.gz.asc
   
   The **gpg --verify** command returns something similar to:
   ```

   ```
gpg: Signature made <DATE_AND_TIME>
gpg: using RSA key <KEY_ID>
gpg: Good signature from "Puppet, Inc. Release Key (Puppet, Inc. Release Key) <release@puppet.com>"
   ```

**Tip:**

If you receive a warning that a valid key path couldn't be found, this means you don't have a trusted path to one of the signatures on the release key.

If you receive a warning that the key is not certified with a trusted signature, this means you haven't told GPG to trust the imported key. Refer to the GPG documentation for more information.

If you received the **Good signature** message, you can proceed to Install PE on page 159.

### Install PE

Installation uses default settings to install PE infrastructure components on a single node. After installing, you can scale or customize your installation as needed.

**Important:** Perform these steps on your target primary server logged in as root. If you're installing on a system that doesn't allow root login, use the command `sudo su -` to switch to the root user.

1. **Download** the tarball appropriate to your operating system and architecture.

   **Tip:** To download packages from the command line, run `wget --content-disposition <URL>` or `curl -JLO <URL>`, using the URL for the tarball you want to download.

2. Unpack the installation tarball:

   ```
tar -xf <TARBALL_FILENAME>
   ```

3. From the installer directory, run the installer and follow the CLI instructions to complete your installation:

   ```
./puppet-enterprise-installer
   ```

4. Optional: Restart the shell in order to use client tool commands.
Configuration parameters and the `pe.conf` file

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

**Tip:** You can install PE using a customized `pe.conf` file by running `./puppet-enterprise-installer -c <PATH_TO_pe.conf>`.

The following are examples of valid parameter and value expressions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FQDNs</td>
<td>&quot;puppet_enterprise::puppet_master_host&quot;: &quot;primary.example.com&quot;</td>
</tr>
<tr>
<td>Strings</td>
<td>&quot;console_admin_password&quot;: &quot;mypassword&quot;</td>
</tr>
<tr>
<td>Arrays</td>
<td>[ &quot;puppet&quot;, &quot;puppetlb-01.example.com&quot; ]</td>
</tr>
<tr>
<td>Booleans</td>
<td>&quot;puppet_enterprise::profile::orchestrator::run_service&quot;: true</td>
</tr>
<tr>
<td></td>
<td>Valid Boolean values are <code>true</code> or <code>false</code> (case sensitive, no quotation marks).</td>
</tr>
<tr>
<td>JSON hashes</td>
<td>&quot;puppet_enterprise::profile::orchestrator::java_args&quot;: {&quot;Xmx&quot;: &quot;256m&quot;, &quot;Xms&quot;: &quot;256m&quot;}</td>
</tr>
<tr>
<td>Integer</td>
<td>&quot;puppet_enterprise::profile::console::rbac_session_timeout&quot;: &quot;60&quot;</td>
</tr>
</tbody>
</table>

**Note:** Don't use Yes (y), No (n), 1, or 0.

**Important:** Don't use single quotes on parameter values. Use double quotes as shown in the examples.

**Installation parameters**

These parameters are required for installation.

**Tip:** To simplify installation, you can keep the default value of `%{::trusted.certname}` for your primary server and provide a console administrator password after running the installer.

**`puppet_enterprise::puppet_master_host`**

The FQDN of the node hosting the primary server, for example `primary.example.com`.

Default: `%{::trusted.certname}`

**Database configuration parameters**

These are the default parameters and values supplied for the PE databases.

This list is intended for reference only; don't change or customize these parameters.

**`puppet_enterprise::activity_database_name`**

Name for the activity database.

Default: `pe-activity`
puppet_enterprise::activity_database_read_user
  Activity database user that can perform only read functions.
  Default: pe-activity-read

puppet_enterprise::activity_database_write_user
  Activity database user that can perform only read and write functions.
  Default: pe-activity-write

puppet_enterprise::activity_database_super_user
  Activity database superuser.
  Default: pe-activity

puppet_enterprise::activity_service_migration_db_user
  Activity service database user used for migrations.
  Default: pe-activity

puppet_enterprise::activity_service_regular_db_user
  Activity service database user used for normal operations.
  Default: pe-activity-write

puppet_enterprise::classifier_database_name
  Name for the classifier database.
  Default: pe-classifier

puppet_enterprise::classifier_database_read_user
  Classifier database user that can perform only read functions.
  Default: pe-classifier-read

puppet_enterprise::classifier_database_write_user
  Classifier database user that can perform only read and write functions.
  pe-classifier-write

puppet_enterprise::classifier_database_super_user
  Classifier database superuser.
  pe-classifier

puppet_enterprise::classifier_service_migration_db_user
  Classifier service user used for migrations.
  Default: pe-classifier

puppet_enterprise::classifier_service_regular_db_user
  Classifier service user used for normal operations.
  Default: pe-classifier-write

puppet_enterprise::orchestrator_database_name
  Name for the orchestrator database.
  Default: pe-orchestrator

puppet_enterprise::orchestrator_database_read_user
  Orchestrator database user that can perform only read functions.
  Default: pe-orchestrator-read

puppet_enterprise::orchestrator_database_write_user
  Orchestrator database user that can perform only read and write functions.
Default: pe-orchestrator-write

**puppet_enterprise::orchestrator_database_super_user**

Orchestrator database superuser.

Default: pe-orchestrator

**puppet_enterprise::orchestrator_service_migration_db_user**

Orchestrator service user used for migrations.

Default: pe-orchestrator

**puppet_enterprise::orchestrator_service_regular_db_user**

Orchestrator service user used for normal operations.

Default: pe-orchestrator-write

**puppet_enterprise::puppetdb_database_name**

Name for the PuppetDB database.

Default: pe-puppetdb

**puppet_enterprise::rbac_database_name**

Name for the RBAC database.

Default: pe-rbac

**puppet_enterprise::rbac_database_read_user**

RBAC database user that can perform only read functions.

Default: pe-rbac-read

**puppet_enterprise::rbac_database_write_user**

RBAC database user that can perform only read and write functions.

Default: pe-rbac-write

**puppet_enterprise::rbac_database_super_user**

RBAC database superuser.

Default: pe-rbac

**puppet_enterprise::rbac_service_migration_db_user**

RBAC service user used for migrations.

Default: pe-rbac

**puppet_enterprise::rbac_service_regular_db_user**

RBAC service user used for normal operations.

Default: pe-rbac-write

**External PostgreSQL parameters**

These parameters are required to install an external PostgreSQL instance. Password parameters can be added to standard installations if needed.

**puppet_enterprise::database_host**

Agent certname of the node hosting the database component. Don't use an alt name for this value.

**puppet_enterprise::database_port**

The port that the database is running on.

Default: 5432

**puppet_enterprise::database_ssl**

True or false. For unmanaged PostgreSQL installations don't use SSL security, set this parameter to false.
Default: true

**puppet_enterprise::database_cert_auth**

true or false.

**Important:** For unmanaged PostgreSQL installations don't use SSL security, set this parameter to false.

Default: true

**puppet_enterprise::puppetdb_database_password**

Password for the PuppetDB database user. Must be a string, such as "mypassword".

**puppet_enterprise::classifier_database_password**

Password for the classifier database user. Must be a string, such as "mypassword".

**puppet_enterprise::classifier_service_regular_db_user**

Database user the classifier service uses for normal operations.

Default: pe-classifier

**puppet_enterprise::classifier_service_migration_db_user**

Database user the classifier service uses for migrations.

Default: pe-classifier

**puppet_enterprise::activity_database_password**

Password for the activity database user. Must be a string, such as "mypassword".

**puppet_enterprise::activity_service_regular_db_user**

Database user the activity service uses for normal operations.

Default: "pe-activity"

**puppet_enterprise::activity_service_migration_db_user**

Database user the activity service uses for migrations.

Default: "pe-activity"

**puppet_enterprise::rbac_database_password**

Password for the RBAC database user. Must be a string, such as "mypassword".

**puppet_enterprise::rbac_service_regular_db_user**

Database user the RBAC service uses for normal operations.

Default: "pe-rbac"

**puppet_enterprise::rbac_service_migration_db_user**

Database user the RBAC service uses for migrations.

Default: "pe-rbac"

**puppet_enterprise::orchestrator_database_password**

Password for the orchestrator database user. Must be a string, such as "mypassword".

**puppet_enterprise::orchestrator_service_regular_db_user**

Database user the orchestrator service uses for normal operations.

Default: pe-orchestrator

**puppet_enterprise::orchestrator_service_migration_db_user**

Database user the orchestrator service uses for migrations.

Default: "pe-orchestrator"
Primary server parameters

Use these parameters to configure and tune the primary server.

pe_install::puppet_master_dnsaltnames

An array of strings representing the DNS altnames to be added to the primary server's SSL certificate.

Default: ["puppet"]

pe_install::install::classification::pe_node_group_environment

String indicating the environment that infrastructure nodes are running in. Specify this parameter if you moved your primary server and other infrastructure nodes from the default production environment after install.

With non-default environments, this setting ensures that your configuration settings are backed up.

Default: "production"

puppet_enterprise::ip_version

Accepts either 4 or 6 to specify a preference for IPv4 or IPv6, but this does not restrict the non-preferred option.

The default is 4 (IPv4). You can set it to 6 if you prefer IPv6.

puppet_enterprise::ipv6_only

You can set it to true to force NGINX to listen on IPv6 only.

Default: false (allows both IPv4 and IPv6)

puppet_enterprise::master::recover_configuration::pe_environment

String indicating the environment that infrastructure nodes are running in. Specify this parameter if you moved your primary server and other infrastructure nodes from the default production environment after installation.

With non-default environments, this setting ensures that your configuration settings are backed up.

Default: "production"

puppet_enterprise::profile::certificate_authority

Array of additional certificates to be allowed access to the /certificate_statusAPI endpoint. This list is added to the base certificate list.

puppet_enterprise::profile::master::check_for_updates

Indicates whether to check for updates when the pe-puppetserver service restarts.

The default is true (check for updates). You can set it to false to not check for updates.

puppet_enterprise::profile::master::code_manager_auto_configure

Set to true to automatically configure the Code Manager service; otherwise, set it to false.

puppet_enterprise::profile::master::r10k_remote

String representing the Git URL to be passed to the r10k.yaml file, for example "git@your.git.server.com:puppet/control.git". The URL can be any URL supported by r10k and Git. This parameter is only required if you want r10k configured when you install PE, and you must also specify puppet_enterprise::profile::master::r10k_private_key.

puppet_enterprise::profile::master::r10k_private_key

String representing the local file path on the primary server where the SSH private key can be found and used by r10k, for example "/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519". This parameter is only required if you want r10k configured when you install PE, and you must also specify puppet_enterprise::profile::master::r10k_remote.
Console and console-services parameters
Use these parameters to customize the behavior of the console and console-services. Parameters that begin with puppet Enterprise::profile can be modified from the console itself. See the configuration methods documents for more information on how to change parameters in the console or Hiera.

puppet Enterprise::profile::console::classifier_synchronization_period
Integer representing, in seconds, the classifier synchronization period, which controls how long it takes the node classifier to retrieve classes from the primary server.
Default: 600 (seconds).

puppet Enterprise::profile::console::rbac_failed_attempts_lockout
Integer specifying how many failed login attempts are allowed on an account before that account is revoked.
Default: 10 (attempts).

puppet Enterprise::profile::console::rbac_password_reset_expiration
Integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user so that they can reset their password.
Default: 24 (hours).

puppet Enterprise::profile::console::rbac_session_timeout
Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console.
Default: 60 (minutes).

puppet Enterprise::profile::console::session_maximum_lifetime
Integer representing the maximum allowable period that a console session can be valid. To not expire before the maximum token lifetime, set to 0.
Supported units are "s" (seconds), "m" (minutes), "h" (hours), "d" (days), "y" (years). Units are specified as a single letter following an integer, for example "1d" (1 day). If no units are specified, the integer is treated as seconds.

puppet Enterprise::profile::console::rbac_token_auth_lifetime
A value representing the default authentication lifetime for a token. It cannot exceed the rbac_token_maximum_lifetime. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "1h".

puppet Enterprise::profile::console::rbac_token_maximum_lifetime
A value representing the maximum allowable lifetime for all tokens. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "10y".

puppet Enterprise::profile::console::console_ssl_listen_port
Integer representing the port that the console is available on.
Default: 443.

puppet Enterprise::profile::console::ssl_listen_address
Nginx listen address for the console.
Default: "0.0.0.0"

puppet Enterprise::profile::console::classifier_prune_threshold
An integer representing the number of days to wait before pruning the node classifier database. The node classifier database contains node check-in history if classifier_node_check_in_storage is enabled.
Set the value to 0 to never prune the node classifier database.
Default: 7 (days), but only has data to prune if classifier_node_check_in_storage is true.
puppet_enterprise::profile::console::classifier_node_check_in_storage
   A Boolean specifying whether to create records when nodes check in with the node classifier. These records
describe how nodes match the node groups they're classified into.
   Set to true to enable node check-in storage.
   Set to false to disable node check-in storage.
   Default: false

puppet_enterprise::profile::console::display_local_time
   A Boolean indicating whether to show timestamps in the local time or UTC.
   Set to true to display timestamps in local time with hover text showing the equivalent UTC time.
   Set to false to show timestamps in UTC time with no hover text.
   Default: false

Modify these configuration parameters in Hiera or pe.conf, not the console:

puppet_enterprise::api_port
   SSL port that the node classifier is served on.
   Default: 4433

puppet_enterprise::console_services::no_longer_reporting_cutoff
   Length of time, in seconds, before a node is considered unresponsive.
   Default: 3600 (seconds)

custom_admin_password
   The password to log into the console, for example "myconsolepassword".
   Default: Specified during installation.

Related information
Configure RBAC and token-based authentication settings on page 295
Tune RBAC and token-based authentication settings, like setting the number of failed attempts a user has before they
are locked out of the console or changing the amount of time a token is valid for.

Orchestrator and orchestration services parameters
Use these parameters to configure and tune the orchestrator and orchestration services.

puppet_enterprise::profile::agent::pxp_enabled
   true to enable the Puppet Execution Protocol service, which is required to use the orchestrator and run Puppet
   from the console, or false.
   Default: true

puppet_enterprise::profile::bolt_server::concurrency
   An integer that determines the maximum number of concurrent requests orchestrator can make to bolt-server.
   CAUTION: Do not set a concurrency limit that is higher than the bolt-server limit. This can cause
   timeouts that lead to failed task runs.
   Default: The default value is set to the current value stored for bolt-server.
puppet_enterprise::profile::orchestrator::global_concurrent_compiles
Integer representing how many concurrent compile requests can be outstanding to the primary server, across all orchestrator jobs.
Default: 8 requests

puppet_enterprise::profile::orchestrator::job_prune_threshold
An integer of 2 or greater, which specifies the number of days to retain job reports.
This parameter sets the corresponding parameter job-prune-days-threshold.
While job_prune_threshold itself has no default value, job-prune-days-threshold has a default of 30 (30 days).

puppet_enterprise::profile::orchestrator::pcp_timeout
Integer representing the length of time, in seconds, before timeout when agents attempt to connect to the Puppet Communications Protocol broker in a Puppet run triggered by the orchestrator.
Default: 30 seconds

puppet_enterprise::profile::orchestrator::run_service
true to enable orchestration services, or false.
Default: true

puppet_enterprise::profile::orchestrator::task_concurrency
Integer representing the number of simultaneous task or plan actions that can run at the same time in the orchestrator. All task and plan actions are limited by this concurrency limit regardless of transport type (WinRM, SSH, PCP).
If a task or plan action runs on multiple nodes, each node consumes one action. For example, if a task needs to run on 300 nodes, and your task_concurrency is set to 200, then the task can run on 200 nodes while the remaining 100 nodes wait in queue.
Default: 250 (actions)

puppet_enterprise::pxp_agent::ping_interval
Integer representing the interval, in seconds, between agents' attempts to ping Puppet Communications Protocol brokers.
Default: 120 seconds

puppet_enterprise::pxp_agent::pxp_logfile
String representing the path to the Puppet Execution Protocol agent log file. Change as needed.
Default: /var/log/puppetlabs/pxp-agent/pxp-agent.log (*nix) or C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log (Windows)

Related information
Configuring Puppet orchestrator on page 614
Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

PuppetDB parameters
Use these parameters to configure and tune PuppetDB.

puppet_enterprise::puppetdb::command_processing_threads
Integer representing how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.
Default: Half the number of cores in your system, for example 8.

puppet_enterprise::profile::master::puppetdb_report_processor_ensure
present to generate agent run reports and submit them to PuppetDB, or absent
Default: present

**puppet_enterprise::puppetdb_port**

Integer in brackets representing the SSL port that PuppetDB listens on.

Default: 8081

**puppet_enterprise::profile::puppetdb::node_purge_ttl**

“Time-to-live” value before deactivated or expired nodes are deleted, along with all facts, catalogs, and reports for the node. For example, a value of "14d" sets the time-to-live to 14 days.

Default: "14d"

**Java parameters**

Use these parameters to configure and tune Java.

**puppet_enterprise::profile::master::java_args**

JVM (Java Virtual Machine) memory, specified as a JSON hash, that is allocated to the Puppet Server service, for example {"Xmx": "4096m", "Xms": "4096m"}.

**puppet_enterprise::profile::puppetdb::java_args**

JVM memory, specified as a JSON hash, that is allocated to the PuppetDB service, for example {"Xmx": "512m", "Xms": "512m"}.

**puppet_enterprise::profile::console::java_args**

JVM memory, specified as a JSON hash, that is allocated to console services, for example {"Xmx": "512m", "Xms": "512m"}.

**puppet_enterprise::profile::orchestrator::java_args**

JVM memory, set as a JSON hash, that is allocated to orchestration services, for example, {"Xmx": "256m", "Xms": "256m"}.

---

**Installing PE in cloud environments**

To install cloud PE, launch a marketplace image from your chosen cloud provider.

**Related information**

Cloud PE on page 17

Puppet Enterprise 2019.8 cloud images are available from several providers.

**Launch an AWS image**

You can launch a cloud PE image from the AWS console or any available AWS SDKs or third-party tools.

1. Launch a PE image and specify details for your deployment.
   - **EC2 instance type** – The image size used for your deployment. For recommendations, see Hardware requirements for standard installations on page 182.
   - **EC2 VPC and subnet** – The VPC or subnet in which to deploy your image.
   - **EC2 security group** – The security group policy to use for your deployment.

   To control access to the image, a new key pair is created.

2. Connect to the image using your new key pair and the username puppetadmin:

   ```bash
   ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
   ```

   SSH keys are automatically provisioned, and no password is required.
3. Wait for the image to boot and for PE configuration to complete. To track the progress of these events: You can use these commands to track progress and troubleshoot issues that might arise.
   a) Switch to a root login shell: `sudo su -`
   b) Run the `check_status.sh` script to track progress:
      ```bash
      /opt/puppetlabs/cloud/bin/check_status.sh --wait
      ```
   c) If the status check times out or PE configuration doesn't execute, run the `configure-pe.rb` script:
      ```bash
      /opt/puppetlabs/cloud/bin/configure-pe.rb
      ```

4. You can't access the PE console until you set a console admin password. To do this:
   a) Switch to a root login shell, if you haven't done so already: `sudo su -`
   b) Run:
      ```bash
      /opt/puppetlabs/puppet/bin/puppet infrastructure console_password
      ```

5. Using a web browser, connect to the console at `https://<PRIMARY_HOSTNAME>`, accept the console's certificate, and login with the username `admin` and the password you specified during installation.

   **Tip:** The console uses an SSL certificate created by your own local Puppet certificate authority. Because this authority is specific to your site, web browsers don't know it or trust it, and you must add a security exception in order to access the console.

   The console indicates that your primary server is actively managed by showing `1 Nodes run in enforcement`.

   Your primary server is now ready to manage nodes.

**Launch an Azure image**

You can launch a cloud PE image from the Azure Portal, Powershell, or SDKs from Ruby, Python, Go, or Java.

1. Launch a PE image and specify details for your deployment.
   - **Resource Group** – Creates a new resource group or reuses an existing group.
   - **Location** – The location for the resource group. If you use an existing resource group, the location must match the resource group's location.
   - **Admin Password** – The password for the admin user. If you select SSH authentication, the password you specify is used as a backup authentication method.
   - **VM Size** – The size used for your deployment. For recommendations, see Hardware requirements for standard installations on page 182.
   - **Admin User Name** – The username to use when logging in with SSH.
   - **Authentication Type** – `sshPublicKey` or `password`
   - **SSH Public Key** – Your SSH public key, if you select SSH as your authentication type.
   - **Virtual Network New Or Existing** – Creates a new virtual network or uses an existing network. If you select existing, you don't need to enter an address prefix or subnet.
   - **Public IP Address New Or Existing** – new to specify a static IP address. If the IP is dynamic and the VM is restarted, you won't be able to access the console, because the console uses the initial public IP address.
   - **Public IP Address Domain Name Label** – The prefix of the VM's fully qualified domain name.
   - **Storage Account New Or Existing** – Creates a new storage account or uses an existing account. If you select existing, you don't need to enter an account type.

   To control access to the image, a new key pair is created.
2. Connect to the image using your new key pair and the username `puppetadmin`:

```bash
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

SSH keys are automatically provisioned, and no password is required.

3. Wait for the image to boot and for PE configuration to complete. To track the progress of these events: You can use these commands to track progress and troubleshoot issues that might arise.
   a) Switch to a root login shell: `sudo su -`
   b) Run the `check_status.sh` script to track progress:

```
/opt/puppetlabs/cloud/bin/check_status.sh --wait
```
   c) If the status check times out or PE configuration doesn't execute, run the `configure-pe.rb` script:

```
/opt/puppetlabs/cloud/bin/configure-pe.rb
```

4. You can't access the PE console until you set a console admin password. To do this:
   a) Switch to a root login shell, if you haven't done so already: `sudo su -`
   b) Run:

```
/opt/puppetlabs/puppet/bin/puppet infrastructure console_password
```

5. Using a web browser, connect to the console at `https://<PRIMARY_HOSTNAME>`, accept the console's certificate, and login with the username `admin` and the password you specified during installation.

   **Tip:** The console uses an SSL certificate created by your own local Puppet certificate authority. Because this authority is specific to your site, web browsers don't know it or trust it, and you must add a security exception in order to access the console.

   The console indicates that your primary server is actively managed by showing `1 Nodes run in enforcement`.

   Your primary server is now ready to manage nodes.

**Launch an Oracle image**

You can launch a cloud PE image from the Oracle Cloud Marketplace console or any available SDKs or third-party tools.

1. Launch a PE image and specify details for your deployment.
   - **Shape** – The image size used for your deployment. For recommendations, see [Hardware requirements for standard installations](#) on page 182.
   - **VCN** – The Virtual Cloud Network or subnet in which to deploy your image.
   - **Network Security Groups** – The security group policy to use for your deployment. These must have all the standard ports PE requires so that services like the console and Puppet Server work correctly.

2. Use the Oracle Cloud console to get an SSH key pair using your chosen method and save it.

3. Connect to the image using your new key pair and the username `puppetadmin`:

```bash
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

SSH keys are automatically provisioned, and no password is required.

© 2022 Puppet, Inc., a Perforce company
4. Wait for the image to boot and for PE configuration to complete. To track the progress of these events: You can use these commands to track progress and troubleshoot issues that might arise.
   a) Switch to a root login shell: `sudo su -`
   b) Run the `check_status.sh` script to track progress:
      ```bash
      /opt/puppetlabs/cloud/bin/check_status.sh --wait
      ```
   c) If the status check times out or PE configuration doesn't execute, run the `configure-pe.rb` script:
      ```bash
      /opt/puppetlabs/cloud/bin/configure-pe.rb
      ```

5. You can't access the PE console until you set a console admin password. To do this:
   a) Switch to a root login shell, if you haven't done so already: `sudo su -`
   b) Run:
      ```bash
      /opt/puppetlabs/puppet/bin/puppet infrastructure console_password
      ```

6. Using a web browser, connect to the console at `https://<PRIMARY_HOSTNAME>`, accept the console's certificate, and login with the username `admin` and the password you specified during installation.

   **Tip:** The console uses an SSL certificate created by your own local Puppet certificate authority. Because this authority is specific to your site, web browsers don't know it or trust it, and you must add a security exception in order to access the console.

   The console indicates that your primary server is actively managed by showing `1 Nodes run in enforcement`.

   Your primary server is now ready to manage nodes.

### Purchasing and installing a license key

Your license must support the number of nodes that you want to manage with Puppet Enterprise.

#### Complimentary license

You can manage up to 10 nodes at no charge, and no license key is needed. When you have 11 or more active nodes and no license key, license warnings appear in the console until you install an appropriate license key.

#### Purchased license

To manage 11 or more active nodes, you must purchase a license. After you purchase a license and install a license key file, your licensed node count and subscription expiration date appear on the **License** page.

**Note:** To support spikes in node usage, four days per calendar month you can exceed your licensed node count up to double the number of nodes you purchased. This increased number of nodes is called your **bursting limit**. You must buy more nodes for your license if you exceed the licensed node count within the bursting limit on more than four days per calendar month, or if you exceed your bursting limit at all. In these cases, license warnings appear in the console until you contact your Puppet representative.

**Related information**

- How nodes are counted on page 457
Your node count is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your bursting limit. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

Getting a license

Contact our sales team to purchase a new license, or to upgrade, renew, or add nodes.

Tip: To reduce your active node count and free up licenses, remove inactive nodes from your deployment. By default, nodes with Puppet agents are automatically deactivated after seven days with no activity (no new facts, catalog, or reports).

Related information

Remove agent nodes on page 453
If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Install a license key

Install the license.key file to upgrade from a test installation to an active installation.

1. Install the license key by copying the file to /etc/puppetlabs/license.key on the primary server node.
2. Verify that Puppet has permission to read the license key by checking its ownership and permissions: ls -la /etc/puppetlabs/license.key
3. If the ownership is not root and permissions are not -rw-r--r-- (octal 644), set them:

   sudo chown root:root /etc/puppetlabs/license.key
   sudo chmod 644 /etc/puppetlabs/license.key

View license details for your environment

Check the number of active nodes in your deployment, the number of licensed nodes you purchased, and the expiration date for your license.

Procedure

• In the console, click License.

The License page opens with information on your licensed nodes, bursting limit, and subscription expiration date. Any license warnings that appear in the console navigation are explained here. For example, if your license is expired or out of compliance.

Related information

Usage endpoints on page 755
Use the /usage endpoint to view details about the active nodes in your deployment.

Remove agent nodes on page 453
If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Installing agents

You can install Puppet Enterprise agents on *nix, Windows, and macOS.

To install FIPS-enabled PE agents, install the appropriate FIPS-enabled agent on a third party supported platform with FIPS mode enabled. You can use FIPS-enabled agents with a non-FIPS enabled primary server.

Agents are typically installed from a package repository hosted on your primary server. The PE package management repository is created during installation of the primary server and serves packages over HTTPS using the same port as the primary server (8140). This means agents don't require any new ports to be open other than the one they already need to communicate with the primary server.
Agent packages can be found on the primary server in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/`. This directory contains the platform specific repository file structure for agent packages. For example, if your primary server is running on CentOS 7, in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/`, there's a directory `el-7-x86_64`, which contains the directories with all the packages needed to install an agent.

After installing agents, you must sign their certificates.

**Using the install script**

The install script installs and configures the agent on target nodes using installation packages from the PE package management repo.

The agent install script performs these actions:

- Detects the operating system on which it's running, sets up an APT, YUM, or zipper repo that refers back to the primary server, and then pulls down and installs the `puppet-agent` packages. If the install script can't find agent packages corresponding to the agent's platform, it fails with an error telling you which `pe_repo` class you need to add to the primary server.
- Downloads a tarball of plugins from the primary server. This feature is controlled by the settings `pe_repo::enable_bulk_pluginsync` and `pe_repo::enable_windows_bulk_pluginsync`, which is set to `true` (enabled) by default. Depending on how many modules you have installed, bulk plug-in sync can speed agent installation significantly.

  **Note:** If your primary server runs in a different environment from your agent nodes, you might see some reduced benefit from bulk plug-in sync. The plug-in tarball is created based on the plug-ins running on the primary server agent, which might not match the plug-ins required for agents in a different environment.

- Creates a basic `puppet.conf` file.
- Kicks off a Puppet run.

Automatic downloading of agent installer packages and plugins using the `pe_repo` class requires an internet connection.

  **Tip:** If your primary server uses a proxy server to access the internet, prior to installation, specify `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` in `pe.conf`, Hiera, or in the console, in the `pe_repo` class of the **PE Master** node group.
You can customize agent installation by providing as flags to the script any number of these options, in any order:

<table>
<thead>
<tr>
<th>Option</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| puppet.conf settings         | main:certname=node1.corp.net  
|                              | agent:splay=true  
|                              | agent:environment=development | The puppet.conf file looks like this:  
|                              | [main]  
|                              | certname =  
|                              | node1.corp.net  
|                              | [agent]  
|                              | splay = true  
|                              | environment =  
|                              | development |  
| CSR attribute settings       | extension_requests:pp_role=webserver  
|                              | custom_attributes:challengePassword=abc123 | The installer creates a csr_attributes.yaml file before installing with this content:  
|                              | ---  
|                              | custom_attributes:  
|                              | challengePassword:  
|                              | abc123  
|                              | extension_requests:  
|                              | pp_role: webserver |  
| MSI properties (Windows only) | -PuppetAgentAccountUser  
|                              | ‘pup_adm’  
|                              | PuppetAgentAccountPassword  
|                              | ‘secret’ | The Puppet service runs as pup_adm with a password of secret. |
| Puppet service status        | *nix:  
|                              | --puppet-service-ensure  
|                              | stopped  
|                              | --puppet-service-enable  
|                              | false | The Puppet service is stopped and doesn't boot after installation. An initial Puppet run doesn't occur after installation.  
|                              | Windows  
|                              | -PuppetServiceEnsure  
|                              | stopped  
|                              | -PuppetServiceEnable  
|                              | false |  

© 2022 Puppet, Inc., a Perforce company
**puppet.conf settings**

You can specify any agent configuration option using the install script. Configuration settings are added to puppet.conf.

These are the most commonly specified agent config options:

- server
- certname
- environment
- splay
- splaylimit
- noop

**Tip:** On Enterprise Linux systems, if you have a proxy between the agent and the primary server, you can specify http_proxy_host, for example `-s agent:http_proxy_host=<PROXY_FQDN>`.

See the Configuration Reference for details.

**CSR attribute settings**

These settings are added to puppet.conf and included in the custom_attributes and extension_requests sections of csr_attributes.yaml.

You can pass as many parameters as needed. Follow the `section:key=value` pattern and leave one space between parameters.

See the csr_attributes.yaml reference for details.

*nix install script with example agent setup and certificate signing parameters:

```bash
uri='https://primary.example.com:8140/packages/current/install.bash'
curl --insecure "$uri" | sudo bash -s main:certname=<CERTNAME OTHER THAN FQDN> custom_attributes:challengePassword=<PASSWORD_FOR_AUTOSIGNER_SCRIPT>
```

**Note:** If you are unable to run the installation script, or install the MSI package with msiexec, you can set CSR attributes by creating a csr_attributes.yaml file in the Puppet confdir (default C:\ProgramData \PuppetLabs\puppet\etc\csr_attributes.yaml) prior to installing the Puppet agent package.

Windows install script with example agent setup and certificate signing parameters:

```powershell
```

**MSI properties (Windows)**

For Windows, you can set these MSI properties, with or without additional agent configuration settings.

<table>
<thead>
<tr>
<th>MSI Property</th>
<th>PowerShell flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLDIR</td>
<td>-InstallDir</td>
</tr>
</tbody>
</table>
Windows install script with MSI properties and agent configuration settings:

```powershell
```

**Puppet service status**

By default, the install script starts the Puppet agent service and kicks off a Puppet run. If you want to manually trigger a Puppet run, or you're using a provisioning system that requires non-default behavior, you can control whether the service is running and enabled.

<table>
<thead>
<tr>
<th>Option</th>
<th>*nix</th>
<th>Windows</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensure</td>
<td>--puppet-service-ensure &lt;VALUE&gt;</td>
<td>-PuppetServiceEnsure &lt;VALUE&gt;</td>
<td>• running</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• stopped</td>
</tr>
<tr>
<td>enable</td>
<td>--puppet-service-enable &lt;VALUE&gt;</td>
<td>-PuppetServiceEnable &lt;VALUE&gt;</td>
<td>• true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• manual (Windows only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• mask</td>
</tr>
</tbody>
</table>

For example:

**nix**

```
uri='https://<PRIMARY_FQDN>:8140/packages/current/install.bash'
curl --insecure "$uri" | sudo bash -s -- --puppet-service-ensure stopped
```

**Windows**

```
```

**Install *nix agents**

You have multiple options for installing *nix agents: from the console, from the command line using PE package management or your own package management, with or without internet access, and more. Chose the installation method that best suits your needs.

**Note:** You must enable TLSv1 to install agents on these platforms:

- AIX
- CentOS 5
Installing

- RHEL 5
- SLES 11
- Solaris 10, 11
- Windows Server 2008 R2

Related information
Enable TLSv1 on page 840
TLSv1 and TLSv1.1 are disabled by default in PE.

Install agents from the console
You can use the console to leverage tasks that install *nix or Windows agents on target nodes.

Before you begin
Add classes to the PE Master node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.
Make sure you have permission to run the appropriate task to install agents on all nodes:
- *nix targets use the task `pe_bootstrap::linux`
- Windows targets use the task `pe_bootstrap::windows`

For Windows targets, this task requires:
- Windows 2008 SP2 or newer
- PowerShell version 3 or higher
- Microsoft .NET Framework 4.5 or higher

1. In the console, on the Nodes page, click Add nodes.
2. Click Install agents.
3. Select a transport method.
   - SSH for *nix targets
   - WinRM for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click Add nodes.
   Tip: Click Installation job started to view the job details for the task.

Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

Related information
Managing certificate signing requests in the console on page 161
A certificate signing request appears in the console on the Certificates page in the Unsigned certificates tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

Managing certificate signing requests on the command line on page 243
You can view, approve, and reject node requests using the command line.

Install *nix agents with PE package management
PE provides its own package management to help you install agents in your infrastructure.

Note: The `<PRIMARY_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.
1. If you're installing an agent with a different OS than the primary server, add the appropriate class for the repo that contains the agent packages.
   a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) On the **Classes** tab, enter `pe_repo` and select the repo class from the list of classes.

   **Note:** The repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

c) Click **Add class**, and commit changes.

d) Run `puppet agent -t` to configure the primary server using the newly assigned class.

   The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.

2. SSH into the node where you want to install the agent, and run the installation command appropriate to your environment.
   - **Curl**

     ```bash
     uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
     curl --insecure "$uri" | sudo bash
     ```

     See **Usage notes for curl examples** for information about forming curl commands.

   - **Tip:** On AIX versions 7.10 and earlier, which don't support the `-k` option, use `--tlsv1` instead. If neither `-k` or `--tlsv1` is supported, you must install using a manually transferred certificate.

   - **RHEL 5 and CentOS 5**

     ```bash
     uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
     curl --insecure --tlsv1 "$uri" | sudo bash
     ```

   - **RHEL 5 and CentOS 5**

     ```bash
     wget -O - -q --no-check-certificate https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash | sudo bash
     ```

   - **Solaris 10**

     ```bash
     sudo export PATH=$PATH:/opt/sfw/bin
     wget -O - -q --no-check-certificate --secure-protocol=TLSv1 https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash | bash
     ```

**Install *nix agents with your own package management**

If you choose not to use PE package management to install agents, you can use your own package management tools.

**Before you begin**

Download the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Configure the package manager on your agent node (YUM, APT) to point to that repo.
3. Install the agent using the command appropriate to your environment.

   • YUM

        sudo yum install puppet-agent

   • APT

        sudo apt-get install puppet-agent

**Install *nix agents using a manually transferred certificate**

If you choose not to or can't use `curl --insecure` to trust the primary server during agent installation, you can manually transfer the primary server CA certificate to any machines you want to install agents on, and then run the installation script against that cert.

1. On the machine that you're installing the agent on, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and transfer `ca.pem` to the `certs` directory you created on the agent node.
3. On the agent node, verify file permissions: `chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem`
4. Run the installation command, using the `--cacert` flag to point to the cert:

        cacert='/etc/puppetlabs/puppet/ssl/certs/ca.pem'
        uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
        curl --cacert "$cacert" "$uri" | sudo bash

See [Usage notes for curl examples](#) for information about forming curl commands.

**Install *nix agents without internet access**

If you don't have access to the internet beyond your infrastructure, you can download the appropriate agent tarball from an internet-connected system and then install using the package management solution of your choice.

**Install *nix agents with PE package management without internet access**

Use PE package management to install agents when you don't have internet access beyond your infrastructure.

**Before you begin**

**Download** the appropriate agent tarball.

1. On your primary server, copy the agent tarball to `/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-<AGENT_VERSION>`, for example `/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-5.5.17/`
2. If you're installing an agent with a different OS than the primary server, add the appropriate class for the repo that contains the agent packages.
   a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) On the **Classes** tab, enter `pe_repo` and select the repo class from the list of classes.

      **Note:** The repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

   c) Click **Add class**, and commit changes.
3. Run Puppet: `puppet agent -t`
4. Follow the steps for **Install *nix agents with PE package management** on page 228.
Install *nix agents with your own package management without internet access

Use your own package management to install agents when you don’t have internet access beyond your infrastructure.

Before you begin

Download the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Disable the PE-hosted repo.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
   b) On the Classes tab, find pe_repo class (as well as any class that begins pe_repo::), and click Remove this class.
   c) Commit changes.

Install *nix agents from compilers using your own package management without internet access

If your infrastructure relies on compilers to install agents, you don’t have to copy the agent package to each compiler. Instead, use the console to specify a path to the agent package on your package management server.

Before you begin

Download the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Set the base_path parameter of the pe_repo class to your package management server.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
   b) On the Classes tab, find the pe_repo class and specify the parameter.
   c) Click Add parameter and commit changes.

Install Windows agents

You have multiple options for installing Windows agents: from the console, from a PowerShell window using PE package management, from the command line using an .msi package, and more. Chose the installation method that best suits your needs.

Note: You must enable TLSv1 to install agents on these platforms:

- AIX
- CentOS 5
- RHEL 5
- SLES 11
- Solaris 10, 11
- Windows Server 2008 R2

Related information

Enable TLSv1 on page 840
TLSv1 and TLSv1.1 are disabled by default in PE.

**Install agents from the console**
You can use the console to leverage tasks that install *nix or Windows agents on target nodes.

### Before you begin

Add classes to the **PE Master** node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.

Make sure you have permission to run the appropriate task to install agents on all nodes:

- *nix targets use the task `pe_bootstrap::linux`
- Windows targets use the task `pe_bootstrap::windows`

For Windows targets, this task requires:

- Windows 2008 SP2 or newer
- PowerShell version 3 or higher
- Microsoft .NET Framework 4.5 or higher

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Install agents**.
3. Select a transport method.
   - **SSH** for *nix targets
   - **WinRM** for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click **Add nodes**.

   **Tip:** Click **Installation job started** to view the job details for the task.

Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

### Related information

- **Managing certificate signing requests in the console** on page 161
  A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.
- **Managing certificate signing requests on the command line** on page 243
  You can view, approve, and reject node requests using the command line.

### Install Windows agents with PE package management

To install a Windows agent with PE package management, you use the `pe_repo` class to distribute an installation package to agents. You can use this method with or without internet access.

#### Before you begin

If your primary server doesn't have internet access, **download** the appropriate agent package and save it on your primary server in the location appropriate for your agent systems:

- 32-bit systems — `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/`
- 64-bit systems — `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/`

You must use PowerShell 2.0 or later to install Windows agents with PE package management.
Note: The `<PRIMARY_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
2. On the Classes tab, type `pe_repo` and select the appropriate repo class from the list of classes.
   - 64-bit (x86_64) — `pe_repo::platform::windows_x86_64`.
   - 32-bit (i386) — `pe_repo::platform::windows_i386`.
3. Click Add class and commit changes.
4. On the primary server, run Puppet to configure the newly assigned class.

   The new repository is created on the primary server at `/opt/puppetlabs/server/data/packages/public/`.<br>

5. On the node, open an administrative PowerShell window, and install:

   ```
   [Net.ServicePointManager]::ServerCertificateValidationCallback = {
   $true};
   $webClient.DownloadFile('https://<PRIMARY_HOSTNAME>:8140/packages/current/install.ps1', 'install.ps1'); .\install.ps1 -v
   ```

   After running the installer, you see the following output, which indicates the agent was successfully installed.

   ```
   Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'
   service { 'puppet':
     ensure => 'running',
     enable => 'true',
   }
   ```

Install Windows agents using a manually transferred certificate

If you need to perform a secure installation on Windows nodes, you can manually transfer the primary server CA certificate to target nodes, and run a specialized installation script against that cert.

1. Transfer the installation script and the CA certificate from your primary server to the node you're installing.

<table>
<thead>
<tr>
<th>File</th>
<th>Location on primary server</th>
<th>Location on target node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation script</td>
<td><code>/opt/puppetlabs/server/data/packages/public</code></td>
<td>Any accessible local directory.</td>
</tr>
<tr>
<td>(install.ps1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
   | CA certificate (ca.pem) | /[etc/puppetlabs/puppet/ssl/certs/\]  | `C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\` |}

2. Run the installation script, using the `-UsePuppetCA` flag: `.\install.ps1 -UsePuppetCA`
**Install Windows agents with the .msi package**

Use the Windows .msi package if you need to specify agent configuration details during installation, or if you need to install Windows agents locally without internet access.

**Before you begin**

Download the .msi package.

**Tip:** To install on nodes that don't have internet access, save the .msi package to the appropriate location for your system:

- 32-bit systems — /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/
- 64-bit systems — /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/

**Install Windows agents with the installer**

Use the MSI installer for a more automated installation process. The installer can configure puppet.conf, create CSR attributes, and configure the agent to talk to your primary server.

1. Run the installer as administrator.
2. When prompted, provide the hostname of your primary server, for example puppet.

**Install Windows agents using msiexec from the command line**

Install the MSI manually from the command line if you need to customize puppet.conf, CSR attributes, or certain agent properties.

On the command line of the node you want to install the agent on, run the install command:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi
```

**Tip:**

- You can specify /l*v install.txt to log the progress of the installation to a file.
- You can set CSR attributes by creating a csr_attributes.yaml file in the Puppet confdir (default C:\ProgramData\PuppetLabs\puppet\etc\csr_attributes.yaml) prior to installing the Puppet agent package.

**MSI properties**

If you install Windows agents from the command line using the .msi package, you can optionally specify these properties.

**Important:** If you set a non-default value for PUPPET_SERVER, PUPPET_CA_SERVER, PUPPET_AGENT_CERTNAME, or PUPPET_AGENT_ENVIRONMENT, the installer replaces the existing value in puppet.conf and re-uses the value at upgrade unless you specify a new value. Therefore, if you've customized these properties, don't change the setting directly in puppet.conf; instead, re-run the installer and set a new value at installation.
<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Setting in <code>pe.conf</code></th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLDIR</td>
<td>Location to install Puppet and its dependencies.</td>
<td>n/a</td>
<td>• 32-bit — C:\Program Files \Puppet Labs \Puppet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 64-bit — C:\Program Files \Puppet Labs \Puppet</td>
</tr>
<tr>
<td>PUPPET_SERVER</td>
<td>Hostname where the primary server can be reached.</td>
<td>server</td>
<td>puppet</td>
</tr>
<tr>
<td>PUPPET_CA_SERVER</td>
<td>Hostname where the CA primary server can be reached, if you're using multiple primary servers and only one of them is acting as the CA.</td>
<td>ca_server</td>
<td>Value of PUPPET_SERVER</td>
</tr>
<tr>
<td>PUPPET_AGENT_CERTNAME</td>
<td>Node's certificate name, and the name it uses when requesting catalogs.</td>
<td>certname</td>
<td>Value of facter fdqn</td>
</tr>
<tr>
<td></td>
<td>For best compatibility, limit the value of certname to lowercase letters, numbers, periods, underscores, and dashes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUPPET_AGENT_ENVIRONMENT</td>
<td>Node's environment.</td>
<td>environment</td>
<td>production</td>
</tr>
</tbody>
</table>

**Note:** If a value for the `environment` variable already exists in puppet.conf, specifying it during installation does not override that value.
<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Setting in <code>pe.conf</code></th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUPPET_AGENT_STARTUP_MODE</td>
<td>Whether and how the agent service is allowed to run. Allowed values are:</td>
<td>n/a</td>
<td>Automatic</td>
</tr>
<tr>
<td></td>
<td>• Automatic — Agent starts up when Windows starts and remains running in the background.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Manual — Agent can be started in the services console or with <code>net start</code> on the command line.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Disabled — Agent is installed but disabled. You must change its startup type in the services console before you can start the service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Definition</td>
<td>Setting in pe.conf</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_USER</td>
<td>Windows user account the agent service uses.</td>
<td>n/a</td>
<td>LocalSystem</td>
</tr>
<tr>
<td></td>
<td>This property is useful if the agent needs to access files on UNC shares, because the default LocalService account can't access these network resources. The user account must already exist, and can be either a local or domain user. The installer allows domain users even if they have not accessed the machine before. The installer grants Logon as Service to the user, and if the user isn’t already a local administrator, the installer adds it to the Administrators group. This property must be combined with PUPPET_AGENT_ACCOUNT_PASSWORD and PUPPET_AGENT_ACCOUNT_DOMAIN.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_PASSWORD</td>
<td>Password for the agent's user account.</td>
<td>n/a</td>
<td>No Value</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_DOMAIN</td>
<td>Domain of the agent's user account.</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>REINSTALLMODE</td>
<td>A default MSI property used to control the behavior of file copies during installation.</td>
<td>n/a</td>
<td>amus as of puppet-agent 1.10.10 and puppet-agent 5.3.4 omus in prior releases</td>
</tr>
</tbody>
</table>

**Important:** If you need to downgrade agents, use REINSTALLMODE=amus when calling msiexec.exe at the command line to prevent removing files that the application needs.

To install the agent with the primary server at puppet.acme.com:

```bash
msiexec /qn /norestart /i puppet.msi PUPPET_SERVER=puppet.acme.com
```
To install the agent to a domain user ExampleCorp\bob:

```shell
msiexec /qn /norestart /i puppet-<VERSION>.msi
PUPPET_AGENT_ACCOUNT_DOMAIN=ExampleCorp PUPPET_AGENT_ACCOUNT_USER=bob
PUPPET_AGENT_ACCOUNT_PASSWORD=password
```

**Windows agent installation details**

Windows nodes can fetch configurations from a primary server and apply manifests locally, and respond to orchestration commands.

After installing a Windows node, the Start Menu contains a Puppet folder with shortcuts for running the agent manually, running Facter, and opening a command prompt for use with Puppet tools.

**Note:** You must run Puppet with elevated privileges. Select **Run as administrator** when opening the command prompt.

The agent runs as a Windows service. By default, the agent fetches and applies configurations every 30 minutes. The agent service can be started and stopped independently using either the service control manager UI or the command line `sc.exe` utility.

Puppet is automatically added to the machine's `PATH` environment variable, so you can open any command line and run `puppet`, `facter` and the other batch files that are in the `bin` directory of the Puppet installation. Items necessary for the Puppet environment are also added to the shell, but only for the duration of execution of each of the particular commands.

The installer includes Ruby, gems, and Facter. If you have existing copies of these applications, such as Ruby, they aren't affected by the re-distributed version included with Puppet.

**Program directory**

Unless overridden during installation, PE and its dependencies are installed in `Program Files at \Puppet Labs\Puppet`.

You can locate the `Program Files` directory using the `PROGRAMFILES` variable or the `PROGRAMFILES (X86)` variable.

The program directory contains these subdirectories.

<table>
<thead>
<tr>
<th>Subdirectory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>scripts for running Puppet and Facter</td>
</tr>
<tr>
<td>facter</td>
<td>Facter source</td>
</tr>
<tr>
<td>hiera</td>
<td>Hiera source</td>
</tr>
<tr>
<td>misc</td>
<td>resources</td>
</tr>
<tr>
<td>puppet</td>
<td>Puppet source</td>
</tr>
<tr>
<td>service</td>
<td>code to run the agent as a service</td>
</tr>
<tr>
<td>sys</td>
<td>Ruby and other tools</td>
</tr>
</tbody>
</table>

**Data directory**

PE stores settings, manifests, and generated data — such as logs and catalogs — in the data directory. The data directory contains two subdirectories for the various components:

- `etc` (the `$confdir`): Contains configuration files, manifests, certificates, and other important files.
- `var` (the `$vardir`): Contains generated data and logs.

© 2022 Puppet, Inc., a Perforce company
When you run Puppet with elevated privileges as intended, the data directory is located in the COMMON_APPDATA.aspx folder. This folder is typically located at C:\ProgramData\PuppetLabs. Because the common app data directory is a system folder, it is hidden by default.

If you run Puppet without elevated privileges, it uses a .puppet directory in the current user's home folder as its data directory, which can result in unexpected settings.

Install macOS agents
You can install macOS agents with PE package management, from Finder, or from the command line.

To install macOS agents with PE package management, follow the steps to Install *nix agents with PE package management on page 228.

Important: For macOS agents, the certname is derived from the name of the machine (such as My-Example-Mac). To prevent installation issues, make sure the name of the node uses lowercases letters. If you don’t want to change your computer’s name, you can enter the agent certname in all lowercase letters when prompted by the installer.

Install macOS agents from Finder
You can use Finder to install the agent on your macOS machine.

Before you begin
Download the appropriate agent tarball.

1. Open the agent package .dmg and click the installer .pkg.
2. Follow prompts in the installer dialog.

You must include the primary server hostname and the agent certname.

Install macOS agents from the command line
You can use the command line to install the agent on a macOS machine.

Before you begin
Download the appropriate agent tarball.

1. SSH into the node as a root or sudo user.
2. Mount the disk image: sudo hdiutil mount <DMGFILE>
   A line appears ending with /Volumes/puppet-agent-VERSION. This directory location is the mount point for the virtual volume created from the disk image.
3. Change to the directory indicated as the mount point in the previous step, for example: cd /Volumes/puppet-agent-VERSION
4. Install the agent package: sudo installer -pkg puppet-agent-installer.pkg -target /
5. Verify the installation: /opt/puppetlabs/bin/puppet --version
6. Configure the agent to connect to the primary server: /opt/puppetlabs/bin/puppet config set server <PRIMARY_HOSTNAME>
7. Configure the agent certname: /opt/puppetlabs/bin/puppet config set certname <AGENT_CERTNAME>

macOS agent installation details
macOS agents include core Puppet functionality, plus platform-specific capabilities like package installation, service management with LaunchD, facts inventory with the System Profiler, and directory services integration.
Install non-root agents

Running agents without root privileges can assist teams using PE to work autonomously.

For example, your infrastructure’s platform might be maintained by one team with root privileges while your infrastructure’s applications are managed by a separate team (or teams) with diminished privileges. If the application team wants to be able to manage its part of the infrastructure independently, they can run Puppet without root privileges.

PE is installed with root privileges, so you need a root user to install and configure non-root access to a primary server. The root user who performs this installation can then set up non-root users on the primary server and any nodes running an agent.

Non-root users can perform a reduced set of management tasks, including configuring settings, configuring Facter external facts, running `puppet agent --test`, and running Puppet with non-privileged cron jobs or a similar scheduling service. Non-root users can also classify nodes by writing or editing manifests in the directories where they have write privileges.

Install non-root *nix agents

**Note:** Unless specified otherwise, perform these steps as a root user or as sudo.

1. Install the agent on each node that you want to operate as a non-root user.
2. Log in to the agent node and add the non-root user:
   ```bash
   sudo puppet resource user <UNIQUE NON-ROOT USERNAME> ensure=present managehome=true
   ```
   **Note:** Each non-root user must have a unique name.

3. Set the non-root user password.
   For example, on most *nix systems: `passwd <USERNAME>`

4. Stop the `puppet` service:
   ```bash
   sudo puppet resource service puppet ensure=stopped enable=false
   ```
   By default, the `puppet` service runs automatically as a root user, so it must be disabled.

5. Disable the Puppet Execution Protocol agent.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Agent group.
   b) On the Classes tab, select the puppet_enterprise::profile::agent class, specify parameters, click Add parameter, and then commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxp_enabled</td>
<td>false</td>
</tr>
</tbody>
</table>

6. Change to the non-root user and generate a certificate signing request:
   ```bash
   sudo puppet agent -t --certname "<UNIQUE NON-ROOT USERNAME.HOSTNAME>" --server "<PRIMARY_HOSTNAME>"
   ```
   **Tip:** If you wish to use `su - <NON-ROOT USERNAME>` to switch between accounts, make sure to use the `-- (-l in some unix variants) argument so that full login privileges are correctly granted. Otherwise you might see permission denied errors when trying to apply a catalog.

7. On the primary server or in the console, approve the certificate signing request.
8. On the agent node as the non-root user, set the node’s certname and the primary server hostname, and then run Puppet.

```bash
sudo puppet config set certname `<UNIQUE NON-ROOT USERNAME.HOSTNAME>` --section agent
sudo puppet config set server `<PRIMARY_HOSTNAME>` --section agent
sudo puppet agent -t
```

The configuration specified in the catalog is applied to the node.

**Tip:** If you see Facter facts being created in the non-root user’s home directory, you have successfully created a functional non-root agent.

To confirm that the non-root agent is correctly configured, verify that:

- The agent can request certificates and apply the catalog from the primary server when a non-root user runs Puppet: `puppet agent -t`
- The agent service is not running: `service puppet status`
- Non-root users can collect existing facts by running `facter` on the agent, and they can define new, external facts.

**Install non-root Windows agents**

**Note:** Unless specified otherwise, perform these steps as a root user or as `sudo`.

1. Install the agent on each node that you want to operate as a non-root user.
2. Log in to the agent node and add the non-root user:

```bash
sudo puppet resource user `<UNIQUE NON-ADMIN USERNAME>` ensure=present
managehome=true password="puppet" groups="Users"
```

**Note:** Each non-root user must have a unique name.

3. Stop the `puppet` service:

```bash
sudo puppet resource service puppet ensure=stopped enable=false
```

By default, the `puppet` service runs automatically as a root user, so it must be disabled.

4. Change to the non-root user and generate a certificate signing request:

```bash
puppet agent -t --certname "<UNIQUE NON-ADMIN USERNAME>" --server "<PRIMARY_HOSTNAME>"
```

**Important:** This Puppet run submits a cert request to the primary server and creates a `.puppet` directory structure in the non-root user’s home directory. If this directory is not created automatically, you must manually create it before continuing.

5. As the non-root user, create a configuration file at `%USERPROFILE%/.puppet/puppet.conf` to specify the agent certname and the hostname of the primary server:

```ini
[main]
certname = `<UNIQUE NON-ADMIN USERNAME.hostname>`
server = `<PRIMARY_HOSTNAME>`
```

6. As the non-root user, submit a cert request: `puppet agent -t`.
7. On the primary server or in the console, approve the certificate signing request.

**Important:** It's possible to sign the root user certificate in order to allow that user to also manage the node. However, this introduces the possibility of unwanted behavior and security issues. For example, if your site.pp has no default node configuration, running the agent as non-admin could lead to unwanted node definitions getting generated using alt hostnames, a potential security issue. If you deploy this scenario, ensure the root and non-root users never try to manage the same resources, have clear-cut node definitions, ensure that classes scope correctly, and so forth.

8. On the agent node as the non-root user, run Puppet: `puppet agent -t`.

The configuration specified in the catalog is applied to the node.

**Non-root user functionality**

Non-root users can use a subset of functionality. Any operation that requires root privileges, such as installing system packages, can't be managed by a non-root agent.

***nix non-root functionality**

On *nix systems, as non-root agent you can enforce these resource types:

- cron (only non-root cron jobs can be viewed or set)
- exec (cannot run as another user or group)
- file (only if the non-root user has read/write privileges)
- notify
- schedule
- ssh_key
- ssh_authorized_key
- service
- augeas

**Note:** When running a cron job as non-root user, using the `-u` flag to set a user with root privileges causes the job to fail, resulting in this error message:

```
Notice: /Stage[main]/Main/Node[nonrootuser]/Cron[illegal_action]/ensure: created must be privileged to use -u
```

You can also inspect these resource types (use `puppet resource <resource type>`):

- host
- mount
- package

**Windows non-root functionality**

On Windows systems as non-admin user, you can enforce these types:

- exec
- file

**Note:** A non-root agent on Windows is extremely limited as compared to non-root *nix. While you can use the above resources, you are limited on usage based on what the agent user has access to do (which isn't much). For instance, you can't create a file/directory in `C:\Windows` unless your user has permission to do so.

You can also inspect these resource types (use `puppet resource <resource type>`):

- host
- package

© 2022 Puppet, Inc., a Perforce company
Managing certificate signing requests

When you install a Puppet agent on a node, the agent automatically submits a certificate signing request (CSR) to the primary server. You must accept this request to bring before the node under PE management can be added to your deployment. This allows Puppet to run on the node and enforce your configuration, which in turn adds node information to PuppetDB and makes the node available throughout the console.

You can approve certificate requests from the PE console or the command line. If DNS altnames are set up for agent nodes, you must approve the CSRs on use the command line interface.

Note: Specific user permissions are required to manage certificate requests:
• To accept or reject CSRs in the console or on the command line, you need the permission **Certificate requests: Accept and reject**.
• To manage certificate requests in the console, you also need the permission **Console: View**.

Related information

**Installing agents** on page 223
You can install Puppet Enterprise agents on *nix, Windows, and macOS.

**Managing certificate signing requests in the console**
A certificate signing request appears in the console on the Certificates page in the Unsigned certificates tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

• To manage requests individually, click **Accept** or **Reject**.
• To manage the entire list of requests, click **Accept All** or **Reject All**. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

Related information

**Running Puppet on demand** on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

**Managing certificate signing requests on the command line**
You can view, approve, and reject node requests using the command line.

To view pending node requests on the command line:

```
$ sudo puppetserver ca list
```

To sign a pending request:

```
$ sudo puppetserver ca sign --certname <NAME>
```

Note: You can use the Puppet Server CA CLI to sign certificates with altnames or auth extensions by default.
Configuring agents

You can add additional configuration to agents by editing `/etc/puppetlabs/puppet/puppet.conf` directly, or by using the `puppet config set` sub-command, which edits `puppet.conf` automatically.

For example, to point the agent at a primary server called `primary.example.com`, run `puppet config set server primary.example.com`. This command adds the setting `server = primary.example.com` to the `[main]` section of `puppet.conf`.

To set the certname for the agent, run `/opt/puppetlabs/bin/puppet config set certname agent.example.com`.

Installing compilers

As your Puppet Enterprise infrastructure scales up to 4,000 nodes and beyond, add load-balanced compilers to your installation to increase the number of agents you can manage.

Each compiler increases capacity by 1,500 to 3,000 nodes, until you exhaust the capacity of PuppetDB or the console.

How compilers work

A single primary server can process requests and compile code for up to 4,000 nodes. When you exceed this scale, expand your infrastructure by adding compilers to share the workload and compile catalogs faster.

**Important:** Compilers must run the same OS major version, platform, and architecture as the primary server.

Compilers act as PCP brokers, conveying messages between the orchestrator and Puppet Execution Protocol (PXP) agents. PXP agents connect to PCP brokers running on compilers over port 8142. Status checks on compilers must be sent to port 8140, using `https://<hostname>[:8140]/status/v1/simple`.

© 2022 Puppet, Inc., a Perforce company
Components and services running on compilers

Compilers typically run Puppet Server and PuppetDB services, as well as a file sync client. Older, legacy-style compilers must be converted in order to add PuppetDB.

When triggered by a web endpoint, file sync takes changes from the working directory on the primary server and deploys the code to a live code directory. File sync then deploys that code to all your compilers. By default, compilers check for code updates every five seconds.

The certificate authority (CA) service is disabled on compilers. A proxy service running on the compiler Puppet Server directs CA requests to the primary server, which hosts the CA in default installations.

Compilers also have:

- The repository for agent installation, `pe_repo`
- The controller profile used with PE client tools
- Puppet Communications Protocol (PCP) brokers to enable orchestrator scale

Logs for compilers are located at `/var/log/puppetlabs/puppetserver/`. 

© 2022 Puppet, Inc., a Perforce company
Logs for PCP brokers on compilers are located at `/var/log/puppetlabs/puppetserver/pcp-broker.log`. Logback configuration for PCP broker logs is part of the Orchestration services settings on page 614.

**Using load balancers with compilers**

When using more than one compiler, a load balancer can help distribute the load between the compilers and provide a level of redundancy.

Specifics on how to configure a load balancer infrastructure falls outside the scope of this document, but examples of how to leverage haproxy for this purpose can be found on the HAproxy module Forge page.

**Calculating load balancing**

PCP brokers run on compilers and connect to PXP agents over port 8142. PCP brokers are built on websockets and require many persistent connections. If you're not using HTTP health checks, we recommend using a round robin or random load balancing algorithm for PXP agent connections to PCP brokers, because PCP brokers don't operate independent of the orchestrator and isolate themselves if they become disconnected. You can check connections with the `/status/v1/simple` endpoint for an error state.

Configure your load balancer to avoid closing long-lived connections that have little traffic. In the HAproxy module, you can set the `timeout tunnel` to 6m because PCP brokers disconnect inactive connections after 6 minutes. You can also configure the `idle-timeout` in the PCP broker trapperkeeper service configuration part of your Orchestration services settings on page 614.

Due to the diverse nature of the network communications between the agent and the primary server, we recommend that you implement a load balancing algorithm that distributes traffic between compilers based on the number of open connections. Load balancers often refer to this strategy as "balancing by least connections."

**Using health checks**

The Puppet REST API exposes a status endpoint that can be leveraged from a load balancer health check to ensure that unhealthy hosts do not receive agent requests from the load balancer.

The primary server service responds to unauthenticated HTTP GET requests issued to `https://<hostname>:8140/status/v1/simple`. The API responds with an HTTP 200 status code if the service is healthy.

If your load balancer doesn't support HTTP health checks, a simpler alternative is to check that the host is listening for TCP connections on port 8140. This ensures that requests aren't forwarded to an unreachable instance of the primary server, but it does not guarantee that a host is pulled out of rotation if it's deemed unhealthy, or if the service listening on port 8140 is not a service related to Puppet.

**Related information**

Firewall configuration for large installations on page 194

These are the port requirements for large installations with compilers.

GET `/status/v1/simple` on page 442

The `/status/v1/simple` returns a status that reflects all services the status service knows about.

Configure settings with Hiera on page 287

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

Load balancing for multi-region installations

If you have load balancers in multiple regions, use a global DNS proximity-based service address.

Tip: For guidance about deploying PE in global, multi-region, or multi-network segment scenarios, see the Multi-region Reference Architectures article.
When using a centralized Puppet deployment with multiple regional proxies or load balancers, create a global DNS proximity-based service address for Puppet and use that to route agents to the appropriate regional load balancer based on their location. Set the global DNS proximity-based address as the compiler pool address in Hiera.

For example, in your common.yaml file:

```
pe_repo::compile_master_pool_address: "<PUPPET-GLOBAL-SERVICE-ADDRESS>"
```

Some suitable global DNS proximity-based service address implementations include:

- BIG-IP DNS
- Route 53 Geolocation routing in AWS
- TCP Proxy Global Load Balancing in GCP
- Traffic Manager in Azure

## Install compilers

Installing a compiler adds the specified node to the **PE Infrastructure Agent** and **PE Compiler** node groups and installs the PuppetDB service on the node.

### Before you begin

The node you want to provision as a compiler must have a Puppet agent installed, or you must be able to connect to a non-agent node with SSH.

To install a FIPS-compliant compiler, install the compiler on a **supported platform** with FIPS mode enabled. The node must be configured with sufficient available entropy or the installation process fails.

**Important:** Contact Support for guidance before installing compilers in multi-region installations. If your primary server and compilers are connected with high-latency links or congested network segments, you might experience better PuppetDB performance with legacy compilers.
1. Configure the agent on infrastructure nodes to connect to the primary server.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Agent > PE Infrastructure Agent group.
   b) If you manage your load balancers with agents, on the Rules tab, pin load balancers to the group.
      Pinning load balancers to the PE Infrastructure Agent group ensures that they communicate directly with the primary server.
   c) On the Classes tab, find the puppet_enterprise::profile::agent class and specify these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage_puppet_conf</td>
<td>Specify true to ensure that your setting for server_list is configured in the expected location and persists through Puppet runs.</td>
</tr>
<tr>
<td>pcp_broker_list</td>
<td>Hostname for your primary server and replica, if you have one. Hostnames must include port 8142, for example &quot;PRIMARY.EXAMPLE.COM:8142&quot;, &quot;REPLICA.EXAMPLE.COM:8142&quot;.</td>
</tr>
<tr>
<td>primary_uris</td>
<td>Hostname for your primary server and replica, if you have one, for example &quot;PRIMARY.EXAMPLE.COM&quot;, &quot;REPLICA.EXAMPLE.COM&quot;. This setting assumes port 8140 unless you specify otherwise with host:port.</td>
</tr>
</tbody>
</table>

   d) Remove any values set for pcp_broker_ws_uris.
   e) Commit changes.
   f) Run Puppet on all agents classified into the PE Infrastructure Agent group.

2. Pin the node that you want to provision to the PE Infrastructure Agent group and run Puppet on the node:
   puppet agent -t

3. On your primary server, logged in as root, run:
   puppet infrastructure provision compiler <COMPILER_FQDN>

You can specify these optional parameters:
- dns-alt-names — Comma-separated list of any alternative names that agents use to connect to compilers. The installation uses puppet by default.

   **Note:** If your puppet.conf file includes a dns_alt_names entry, you must include the dns-alt-names parameter when provisioning your compiler.

- no-dns-alt-names — Prevents the installer from setting the default alternative name, puppet. Use this parameter if you don't allow alternative names (allow-subject-alt-names: false in your ca.conf).

- use-ssh — Enables installing on a node that doesn't have a Puppet agent currently installed. You must be able to connect to the node with SSH. You can pair this flag with additional SSH parameters. Run puppet infrastructure provision --help for details.

Configure compilers to appropriately route communication between your primary server and agent nodes.

**Configure compilers**

Compilers must be configured to appropriately route communication between your primary server and agent nodes.

**Before you begin**

- Install compilers and load balancers.
If you need DNS altnames for your load balancers, add them to the primary server.

Ensure port 8143 is open on the primary server or on any workstations used to run orchestrator jobs.

**Note:** This procedure is not intended for installations with load balancers in multiple locations. For details about configuring compilers in multi-region installations, see Load balancing for multi-region installations on page 246.

1. Configure `pe_repo::compile_master_pool_address` to send agent install requests to the load balancer.

   **Note:** If you are using a single compiler, configure `compile_master_pool_address` with the compiler's fully qualified domain name (FQDN).

   a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) On the **Configuration data** tab, select the **pe_repo** class and specify parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compile_master_pool_address</code></td>
<td>Load balancer hostname.</td>
</tr>
</tbody>
</table>

   c) Click **Add data** and commit changes.

2. Run Puppet on the compiler, and then on the primary server.

3. Configure agents to connect orchestration agents to the load balancer.

   a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent** group.
   b) On the **Classes** tab, find the **puppet_enterprise::profile::agent** class and specify parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>manage_puppet_conf</code></td>
<td>Specify <code>true</code> to ensure that your setting for <code>server_list</code> is configured in the expected location and persists through Puppet runs.</td>
</tr>
<tr>
<td><code>pcp_broker_list</code></td>
<td>Hostname for your load balancers. Hostnames must include port 8142, for example <code>&quot;LOADBALANCER1.EXAMPLE.COM:8142&quot;, &quot;LOADBALANCER2.EXAMPLE.COM:8142&quot;</code>.</td>
</tr>
<tr>
<td><code>primary_uris</code></td>
<td>Hostname for your load balancers, for example <code>&quot;LOADBALANCER1.EXAMPLE.COM&quot;, &quot;LOADBALANCER2.EXAMPLE.COM&quot;</code>. This setting assumes port 8140 unless you specify otherwise with <code>host:port</code>.</td>
</tr>
<tr>
<td><code>server_list</code></td>
<td></td>
</tr>
</tbody>
</table>

   c) Remove any values set for `pcp_broker_ws_uris`.
   d) Commit changes.
   e) Run Puppet on the primary server, then run Puppet on all agents, or install new agents.

   This Puppet run configures PXP agents to connect to the load balancer.

**Related information**

**Firewall configuration for large installations** on page 194

These are the port requirements for large installations with compilers.

**Configure settings with Hiera** on page 287

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.
Convert existing compilers

If you have legacy compilers, you can improve their usability and scalability by adding PuppetDB. In addition to installing the PuppetDB service, converting an existing compiler adds the node to the PE Compiler node group and unpins it from the PE Master node group.

**Before you begin**

Open port 5432 from compilers to your primary server or, in extra-large installations, your PE-PostgreSQL node.

**Important:** Contact Support for guidance before converting compilers in multi-region installations. If your primary server and compilers are connected with high-latency links or congested network segments, you might experience better PuppetDB performance with legacy compilers.

On your primary server logged in as root, run:

```
puppet infrastructure run convert_legacy_compiler
compiler=<COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

**Tip:** To convert all compilers:

```
puppet infrastructure run convert_legacy_compiler all=true
```

Run `puppet infrastructure tune` on your primary server and adjust tuning for compilers as needed.

Installing client tools

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

The `pe-client-tools` package is included in the PE installation tarball. When you install, the client tools are automatically installed on the same node as the primary server. When you upgrade, client tools are automatically updated on infrastructure nodes and managed nodes, but on unmanaged nodes, you must re-install the version of client tools that matches the PE version you upgraded to.

Client tools versions align with PE versions. For example, if you're running PE 2019.8, use the 2019.8 client tools. In some cases, we might issue patch releases ("x.y.z") for PE or the client tools. You don't need to match patch numbers between PE and the client tools. Only the "x.y" numbers need to match.

**Note:** To see the version of client tools installed on your system, use the command appropriate for your package manager or operating system. For example, on Red Hat: `rpm -q pe-client-tools`.

The package includes client tools for these services:

- **Orchestrator** — Allow you to control the rollout of changes in your infrastructure, and provides the interface to the orchestration service. Tools include `puppet job` and `puppet task`.
- **Puppet access** — Authenticates you to the PE RBAC token-based authentication service so that you can use other capabilities and APIs.
- **Code Manager** — Provides the interface for the Code Manager and file sync services. Tools include `puppet-code`.
- **PuppetDB CLI** — Enables certain operations with PuppetDB, such as building queries and handling exports.

Because you can safely run these tools remotely, you no longer need to SSH into the primary server to execute commands. Your permissions to see information and to take action are controlled by PE role-based access control. Your activity is logged under your username rather than under root or the `pe-puppet` user.

**Related information**

[Orchestrator configuration files](#) on page 617
The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate workstation, there are two types of configuration files: a global configuration file and a user-specified configuration file.

Configure puppet-access on page 347
The puppet-access command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. Ensure puppet-access is configured correctly before using it to generate authentication tokens.

Installing and configuring puppet-code on page 792
PE automatically installs and configures the puppet-code command on your primary server as part of the included PE client tools package. You can also set up puppet-code on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.

Supported PE client tools operating systems
The PE client tools package can be installed on the following platforms.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
<th>Arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentOS</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>macOS</td>
<td>10.15, 11, 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 10.15: x86_64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 11: x86_64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 12: x86_64, M1, M2</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>10, 11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 10: x86, x64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 11: x64</td>
<td></td>
</tr>
<tr>
<td>Oracle Linux</td>
<td>6, 7, 8, 9</td>
<td>x86_64</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>6, 7, 8, 9</td>
<td>x86_64</td>
</tr>
<tr>
<td>Scientific Linux</td>
<td>6, 7, 8, 9</td>
<td>x86_64</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>12, 15</td>
<td>x86_64</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>18.04, 20.04, 22.04</td>
<td>amd64</td>
</tr>
</tbody>
</table>

Install PE client tools on a managed workstation
To use the client tools on a system other than the primary server, where they're installed by default, you can install the tools on a controller node.

Before you begin
Controller nodes must be running the same OS as your primary server and must have an agent installed.

1. In the console, create a controller classification group, for example PE Controller, and ensure that its Parent name is set to All Nodes.
2. Select the controller group and add the puppet_enterprise::profile::controller class.
3. Pin the node that you want to be a controller to the controller group.
   a) In the controller group, on the Rules tab, in the Certname field, enter the certname of the node.
   b) Click Pin node and commit changes.
4. Run Puppet on the controller machine.
Install PE client tools on an unmanaged workstation

You can install the `pe-client-tools` package on any workstation running a supported OS. The workstation OS does not need to match the primary server OS.

**Before you begin**
Review prerequisites for timekeeping, name resolution, and firewall configuration, and ensure that these ports are available on the workstation.

- **8143** — The orchestrator client uses this port to communicate with orchestration services running on the primary server.
- **4433** — The Puppet access client uses this port to communicate with the RBAC service running on the primary server.
- **8170** — If you use the Code Manager service, it requires this port.

Install PE client tools on an unmanaged Linux workstation

1. On the workstation, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the directory you created on the workstation.
3. On the workstation, make sure file permissions are correct: `chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem`
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary server.
5. Download the `pe-client-tools` package for the platform appropriate to your workstation.

Install PE client tools on an unmanaged Windows workstation

You can install the client tools on a Windows workstation using the setup wizard or the command line.

To start using the client tools on your Windows workstation, open the **PE ClientTools Console** from the **Start** menu.

1. On the workstation, create the directory `C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs`.
   
   For example: `mkdir C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs`
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the directory you created on the workstation.
3. On the workstation, make sure the file permissions are set to read-only for `C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem`.
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary server.
5. Install the client tools using guided setup or the command line.
   • Guided setup
     b. Double-click the pe-client-tools .msi file.
     c. Follow prompts to accept the license agreement and select the installation location.
     d. Click Install.
   • Command line
     b. From the command line, run the installer:

     ```
     msiexec /i <PATH TO PE-CLIENT-TOOLS.MSI> TARGETDIR="<INSTALLATION DIRECTORY>"
     ```

     TARGETDIR is optional.

**Install PE client tools on an unmanaged macOS workstation**
You can install the client tools on a macOS workstation using Finder or the command line.

1. On the workstation, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the
   directory you created on the workstation.
3. On the workstation, make sure file permissions are correct: `chmod 444 /etc/puppetlabs/puppet/
   ssl/certs/ca.pem`
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary
   server.
5. Install the client tools using Finder or the command line.
   • Finder
     a. Download the macOS pe-client-tools-package.
     b. Open the `pe-client-tools .dmg` and click the installer .pkg.
     c. Follow the prompts to install the client tools.
   • Command line
     a. Download the macOS pe-client-tools-package.
     b. Mount the disk image: `sudo hdiutil mount <DMGFILE>`. 
       A line appears ending with `/Volumes/puppet-agent-VERSION`. This directory location is the
       mount point for the virtual volume created from the disk image.
     e. Run `cd -` and then run `sudo umount /Volumes/pe-client-tools-VERSION`.

**Configuring and using PE client tools**
Use configuration files to customize how client tools communicate with the primary server.

For each client tool, you can create config files for individual machines (global) or for individual users. Configuration
files are structured as JSON.

Save configuration files to these locations:
   • Global
     • *nix — `/etc/puppetlabs/client-tools/`
     • Windows — `%ProgramData%\puppetlabs\client-tools`
On managed client nodes where the operating system and architecture match the primary server, you can have PE manage Puppet code and orchestrator global configuration files using the puppet_enterprise::profile::controller class.

For example configuration files and details about using the various client tools, see the documentation for each service.

<table>
<thead>
<tr>
<th>Client tool</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestrator</td>
<td>• How Puppet orchestrator works on page 604</td>
</tr>
<tr>
<td></td>
<td>• Running Puppet on demand from the CLI on page 627</td>
</tr>
<tr>
<td></td>
<td>• Running tasks from the command line on page 641</td>
</tr>
<tr>
<td>Puppet access</td>
<td>• Token-based authentication on page 347</td>
</tr>
<tr>
<td>Puppet code</td>
<td>• Triggering Code Manager on the command line on page 792</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>• PuppetDB CLI</td>
</tr>
</tbody>
</table>

**Related information**

**Orchestrator configuration files** on page 617
The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate workstation, there are two types of configuration files: a global configuration file and a user-specified configuration file.

**Configure puppet-access** on page 347
The puppet-access command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. Ensure puppet-access is configured correctly before using it to generate authentication tokens.

**Installing and configuring puppet-code** on page 792
PE automatically installs and configures the puppet-code command on your primary server as part of the included PE client tools package. You can also set up puppet-code on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.

**Uninstalling**

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

**Uninstall infrastructure nodes**
The puppet-enterprise-uninstaller script is installed on the primary server. In order to uninstall, you must run the uninstaller on each infrastructure node.
By default, the uninstaller removes the software, users, logs, cron jobs, and caches, but it leaves your modules, manifests, certificates, databases, and configuration files in place, as well as the home directories of any users it removes. If you want to uninstall and reinstall a primary server on the same system, see Uninstaller options on page 256 for options that prevent conflicts by removing those files and databases.

1. From the infrastructure node that you want to uninstall, from the command line as root, navigate to the installer directory and run the uninstall command: $ sudo ./puppet-enterprise-uninstaller

   **Note:** If you don't have access to the installer directory, you can run /opt/puppetlabs/bin/puppet-enterprise-uninstaller.

2. Follow prompts to uninstall.

3. (Optional) If you don't uninstall the primary server, and you plan to reinstall on an infrastructure node at a later date, remove the agent certificate for that component from the primary server. On the primary server: puppetserver ca clean --certname <PE COMPONENT CERT NAME>.

## Uninstall agents

You can remove the agent from nodes that you no longer want to manage.

**Note:** Uninstalling the agent doesn't remove the node from your environment. To completely remove all traces of a node, you must also purge the node.

### Related information

Adding and removing agent nodes on page 452

After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.

## Uninstall *nix agents

The *nix agent package includes an uninstall script, which you can use when you're ready to retire a node.

1. On the agent node, run the uninstall script: /opt/puppetlabs/bin/puppet-enterprise-uninstaller

2. Follow prompts to uninstall.

3. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: puppetserver ca clean --certname <AGENT CERT NAME>.

## Uninstall Windows agents

To uninstall the agent from a Windows node, use the Windows Add or Remove Programs interface, or uninstall from the command line.

Uninstalling the agent removes the Puppet program directory, the agent service, and all related registry keys. The data directory remains intact, including all SSL keys. To completely remove Puppet from the system, manually delete the data directory.

1. Use the Windows Add or Remove Programs interface to remove the agent.

   Alternatively, you can uninstall from the command line if you have the original .msi file or know the product code of the installed MSI, for example: msiexec /qn /norestart /x [puppet.msi|<PRODUCT_CODE>]

2. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: puppetserver ca clean --certname <AGENT CERT NAME>.

## Uninstall macOS agents

Use the command line to remove all aspects of the agent from macOS nodes.
1. On the agent node, run these commands:

   ```bash
   rm -rf /var/log/puppetlabs
   rm -rf /var/run/puppetlabs
   pkgutil --forget com.puppetlabs.puppet-agent
   launchctl remove puppet
   rm -rf /Library/LaunchDaemons/com.puppetlabs.puppet.plist
   launchctl remove pxp-agent
   rm -rf /Library/LaunchDaemons/com.puppetlabs.pxp-agent.plist
   rm -rf /etc/puppetlabs
   rm -rf /opt/puppetlabs
   ```

2. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server:

   ```bash
   puppetserver ca clean --certname <AGENT CERT NAME>
   ```

**Uninstaller options**

You can use the following command-line flags to change the uninstaller's behavior.

- `-p` — Purge additional files. With this flag, the uninstaller also removes all configuration files, modules, manifests, certificates, the home directories of any users created by the installer, and the Puppet public GPG key used for package verification.
- `-d` — Also remove any databases created during installation.
- `-h` — Display a help message.
- `-n` — Run in `noop` mode; show commands that would have been run during uninstallation without actually running them.
- `-y` — Don't ask to confirm uninstallation, assuming an answer of yes.

To remove every trace of PE from a system, which is required if you want to reinstall PE on the same system, run:

```bash
$ sudo ./puppet-enterprise-uninstaller -d -p
```

**Upgrading**

To upgrade your Puppet Enterprise deployment, you must upgrade both the infrastructure components and agents.

**CAUTION:**

Major primary server OS upgrades (such as Ubuntu 18.04 to 20.04) require Backing up and restoring PE on page 841.

Major agent OS upgrades require reinstalling the `puppet-agent` package (as explained in Installing agents on page 223) and reinstalling any Ruby plugins/gems that were added at `/opt/puppetlabs/puppet/bin/gem`.
• **Upgrading Puppet Enterprise** on page 257
  Upgrade your PE installation as new versions become available.
• **Upgrading agents** on page 273
  Upgrade your agents as new versions of Puppet Enterprise become available. The puppet_agent module helps automate upgrades, and provides the safest upgrade. Alternatively, you can upgrade individual nodes using a script.

---

## Upgrading Puppet Enterprise

Upgrade your PE installation as new versions become available.

### Upgrade paths

These are the valid upgrade paths for PE.

<table>
<thead>
<tr>
<th>If you're on version...</th>
<th>Upgrade to...</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.8.z</td>
<td>2019.8.12 (overlap support) or 2019.7.z (LTS)</td>
<td><strong>Important:</strong> 2019.8.z is in overlap support until EOL in February 2023. You can upgrade to the new LTS (2019.7.z) from any 2019.8.z version. For important information about this upgrade, refer to <a href="#">Upgrading Puppet Enterprise in the 2021.7.z documentation</a>.</td>
</tr>
<tr>
<td>2019.y</td>
<td>2019.8.z</td>
<td></td>
</tr>
<tr>
<td>2018.1.2 or later</td>
<td>2019.8.z</td>
<td>You must have version 2018.1.2 or later in order to complete prerequisites for upgrade to the latest version.</td>
</tr>
<tr>
<td>2018.1.3 or later with disaster recovery</td>
<td>2019.8.z</td>
<td></td>
</tr>
<tr>
<td>2018.1.0 or 2018.1.1</td>
<td>2018.1.z</td>
<td>With disaster recovery enabled, you must have version 2018.1.3 in order to upgrade to the latest version. Alternatively, you can forget and then recreate your replica after upgrade.</td>
</tr>
<tr>
<td>2017.3.z</td>
<td>2018.1.z</td>
<td></td>
</tr>
<tr>
<td>2017.2.z</td>
<td>2018.1.z</td>
<td></td>
</tr>
<tr>
<td>2017.1.z</td>
<td>2018.1.z</td>
<td></td>
</tr>
<tr>
<td>2016.5.z</td>
<td>2018.1.z</td>
<td></td>
</tr>
<tr>
<td>2016.4.10 or later</td>
<td>2016.4.z, then 2018.1</td>
<td>To upgrade to 2018.1 from 2015.2.z through 2016.4.9, you must first upgrade to the latest 2016.4.z.</td>
</tr>
<tr>
<td>2016.4.9 or earlier</td>
<td>2016.4.z, then 2018.1</td>
<td></td>
</tr>
<tr>
<td>2016.2.z</td>
<td>2016.4.z, then 2018.1</td>
<td></td>
</tr>
<tr>
<td>2016.1.z</td>
<td>2016.4.z, then 2018.1</td>
<td></td>
</tr>
<tr>
<td>If you're on version...</td>
<td>Upgrade to...</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>2015.3.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015.2.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8.x</td>
<td>2016.4.z, then 2018.1</td>
<td>To upgrade from 3.8.x, you must first migrate to the latest 2016.4.z. This upgrade requires a different process than upgrades from other versions.</td>
</tr>
</tbody>
</table>

**Related information**

Component versions in recent PE releases on page 14

These tables show which components are in Puppet Enterprise (PE) 2019.y and 2018.y. releases. Component version tables for prior releases are available in the Documentation for other PE versions on page 30. Component version tables for the current STS and LTS releases are available in the latest PE documentation.

Primary server and agent compatibility on page 16

Use this table to verify that you’re using a compatible version of the agent for your PE or Puppet server.

**Upgrade cautions**

These are the major changes to PE since the last long-term support release, 2018.1. Review these recommendations and plan accordingly before upgrading to this version.

**Important:** Always back up your installation before performing any upgrade.

Retired primary server platforms in 2019.8

Support for Enterprise Linux 6 and Ubuntu 16.04 as a primary server platform was removed in 2019.8. If your primary server is installed on one of these platforms, you must update the operating system before you can upgrade to this version of PE.

Follow these steps to upgrade from an unsupported primary server platform.

1. Configure a new node with a supported primary server platform, for example Enterprise Linux 7 or 8, or Ubuntu 18.04.
2. Install your current PE version on the new node.
3. Back up your existing installation.
4. Restore your installation on the new primary server using the backup you created.
5. Upgrade to the latest PE version.
6. If you have compilers, reprovision them.
7. If you have a replica, forget and then reprovision it.

**Related information**

Back up your infrastructure on page 842

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7 on page 259

Deleting PuppetDB reports and truncating the resource events table before you upgrade can reduce migration time and lessen downtime, especially when the upgrade involves a significant database migration.

Install PE on page 159

Installation uses default settings to install PE infrastructure components on a single node. After installing, you can scale or customize your installation as needed.

Restore your infrastructure on page 842

Use the restore commands to migrate your primary server to a new host or to recover from system failure.

Upgrade PE on page 263
Upgrade PE infrastructure components to get the latest features and fixes. Follow the upgrade instructions for your installation type to ensure you upgrade components in the correct order. Coordinate upgrades to ensure all infrastructure nodes are upgraded in a timely manner, because agent runs and replication fail on infrastructure nodes running a different agent version than the primary server.

Install compilers on page 247

Installing a compiler adds the specified node to the PE Infrastructure Agent and PE Compiler node groups and installs the PuppetDB service on the node.

Forget a replica on page 328

Forgetting a replica cleans up classification and database state, preventing degraded performance over time.

Provision and enable a replica on page 325

Provisioning a replica duplicates specific components and services from the primary server to the replica. Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.


Deleting PuppetDB reports and truncating the resource events table before you upgrade can reduce migration time and lessen downtime, especially when the upgrade involves a significant database migration.

CAUTION: If you have a replica, you must stop PuppetDB from syncing the deleted reports back while you are deleting the reports. **Therefore, it is crucial that you delete reports on your primary server and replica at the same time.** Make sure the report deletion process is complete on both, and then restart the PuppetDB service on your primary server and replica.

1. On your primary server (and replica), stop the PuppetDB service by running:

   ```bash
   puppet resource service pe-puppetdb ensure=stopped
   ```

   **Remember:** If you have a replica, you **must** do these steps on the primary server and replica at the same time.

2. The next step depends on the Puppet Enterprise (PE) version you’re upgrading from. Regardless of the version, if you have a replica, you must perform these steps on the primary server and replica at the same time.

   a. **If you're upgrading from PE 2019.7 or later:** Run the packaged `delete-reports` command on your primary server (and replica):

   ```bash
   /opt/puppetlabs/bin/puppetdb delete-reports
   ```

   b. **If you're upgrading from a PE version earlier than 2019.7:**

   1. On your PE-PostgreSQL server, create a file named `/tmp/delete-reports.sql` and set it to be owned by the `pe-postgres` user (`chown pe-postgres:pe-postgres /tmp/delete-reports.sql`). Then,

   2. Add contents to the `.sql` file according to your PE version.

   For PE versions 2019.3 through 2019.6, insert the following content:

   ```sql
   BEGIN TRANSACTION;
   ALTER TABLE certnames DROP CONSTRAINT IF EXISTS certnames_reports_id_fkey;
   UPDATE certnames SET latest_report_id = NULL;
   DO $$ DECLARE
   r RECORD;
   BEGIN
   FOR r IN (SELECT tablename FROM pg_tables
   WHERE tablename LIKE 'resource_events_%') LOOP
   EXECUTE 'DROP TABLE ' || quote_ident(r.tablename);
   ```

   © 2022 Puppet, Inc., a Perforce company
END LOOP;
END $$;

TRUNCATE TABLE reports CASCADE;

ALTER TABLE certnames
ADD CONSTRAINT certnames_reports_id_fkey
FOREIGN KEY (latest_report_id) REFERENCES
reports(id) ON DELETE SET NULL;

COMMIT TRANSACTION;

For PE versions earlier than 2019.3, insert the following content:

BEGIN TRANSACTION;

ALTER TABLE certnames DROP CONSTRAINT IF
EXISTS certnames_reports_id_fkey;
UPDATE certnames SET latest_report_id =
NULL;

TRUNCATE TABLE reports CASCADE;

ALTER TABLE certnames
ADD CONSTRAINT certnames_reports_id_fkey
FOREIGN KEY (latest_report_id) REFERENCES
reports(id) ON DELETE SET NULL;

COMMIT TRANSACTION;

3. Run this command on your primary server (and replica):

```
su - pe-postgres -s /bin/bash -c "/opt/puppetlabs/server/bin/psql -d pe-puppetdb -f /tmp/delete-reports.sql"
```

3. Wait for the deletion process to finish.
4. Restart the PuppetDB service on your primary server (and replica) by running:

```
puppet resource service pe-puppetdb ensure=running
```

**Java 11 upgrade in PE 2019.3**

PE 2019.3 includes an upgrade from Java version 8 to version 11. If you've customized PE Java services, or use plug-ins that include Java code, test PE 2019.3 and later thoroughly in a non-production environment before upgrading.

**Retired split installations in PE 2019.2**

Split installations, where the primary server, console, and PuppetDB are installed on separate nodes, are no longer supported.

Before upgrading to 2019.2 or later, you must migrate from a split to a standard installation. For instructions, see [Migrate from a split to a standard installation](#) in the documentation for your current version.

**Orchestrator memory use increase in PE 2019.2**

Puppet orchestrator uses more memory in version 2019.2 than in previous versions due to the addition of a Java virtual machine (JVM), which enables new features and functionalities such as plans. If your memory use is near capacity when running PE 2019.1 or older versions, allocate additional memory before upgrading to PE 2019.2.

Additionally, take care when writing plans, as they can require more memory than is allocated to the orchestrator. To work around this issue, rewrite the plan or increase the memory allocated to the orchestrator.
**PostgreSQL 11 upgrade in PE 2019.2**

PE 2019.2.0 includes an upgrade from `pe-postgresql` version 9.6 to version 11. This upgrade involves a datastore migration that requires extra disk space (110% of the current 9.6 datastore) and extra time to upgrade (roughly two and four minutes of additional time per 10GB of datastore size). The installer issues a warning and cancels the upgrade if there is insufficient space.

To review the size of your PostgreSQL installation as well as the size and number of available bytes for the partition, run `facter -p pe_postgresql_info` on the node that runs the `pe-postgresql` service.

To speed the migration and optimize queries, clean up the PE-PostgreSQL database prior to upgrade by applying the `pe_databases` module to nodes running the `pe-postgresql` service. For best results, apply the module at least one week prior to upgrade to allow the module's maintenance schedule enough time to clean all databases.

After upgrading, you can optionally remove packages and directories associated with older PostgreSQL versions with the command `puppet infrastructure run remove_old_postgresql_versions`. If applicable, the installer prompts you to complete this cleanup.

**Note:** If you use an external PostgreSQL instance not managed by PE, you must separately upgrade the instance to PostgreSQL 11. For more information, see Upgrading PostgreSQL.

**TLSv1 and v1.1 disabled in PE 2019.1**

As of PE 2019.1, TLSv1 and TLSv1.1 is now disabled by default to comply with security regulations. You must enable TLSv1 to upgrade agents on these platforms:

- AIX
- CentOS 5
- RHEL 5
- SLES 11
- Solaris 10, 11
- Windows Server 2008 R2

**Certificate architecture and handling in PE 2019.0**

PE 2019.0 and later, courtesy of Puppet Server, uses an intermediate certificate authority architecture by default. When you upgrade to PE 2019.0 or later, you have several options for whether and when to adopt the new intermediate certificate architecture:

- **To upgrade to 2019.0 or later and keep your existing CA**, upgrade infrastructure nodes and agents as normal. You can continue to use pre-6.x agents with a Puppet 6.x or PE 2019.0 or later primary server as long as you don't regenerate certificates.
- **To migrate to 2019.0 or later and keep your existing CA**, install the new version and copy `/etc/puppetlabs/puppet/ssl` from your old primary server. You can continue to use pre-6.x agents with a Puppet 6.x or PE 2019.0 or later primary server as long as you don't regenerate certificates.
- **To adopt the new CA architecture**, upgrade both your primary server and agents, and then regenerate certificates. If you don't upgrade all of your nodes to 6.x, don't regenerate certificates, because pre-6.x agents won't work with the new CA architecture.

**MCollective removal in PE 2019.0**

If you're upgrading from a 2018.1 installation with MCollective enabled, you must take additional steps to ensure a successful upgrade. If any nodes are configured with MCollective or ActiveMQ profiles when you attempt to upgrade, the installer halts and prompts you to remove the profiles.

**Before upgrading to 2019.x**
• Remove MCollective from nodes in your infrastructure:
  1. In the console, click Node groups, and select the node group PE Infrastructure.
  2. On the Configuration tab, in the puppet_enterprise class, set the mcollective parameter to absent.
  3. Click Add parameter and commit the change, then run Puppet on infrastructure nodes.

After completing these steps, the server components of MCollective, including pe-activemq and the peadmin user, are removed from the primary server and the MCollective service on agents is stopped. You must complete the upgrade to 2019.0 or later to completely remove MCollective from agents.

• Remove any of these deprecated parameters:
  • mcollective Middleware Hosts
  • mcollective
  • mcollective Middleware Port
  • mcollective Middleware User
  • mcollective Middleware Password

Tip: If your PuppetDB includes outdated catalogs for nodes that aren’t currently being managed, the installer might report that MCollective is active on those nodes. You can deactivate the nodes with puppet node deactivate or use Puppet to update the records.

After upgrading to 2019.x

• Manually remove these node groups:
  • PE MCollective
  • PE ActiveMQ Broker
  • Any custom node group you created for ActiveMQ hubs
  • If you customized classification with references to MCollective or ActiveMQ profiles, remove the profiles from your classification. In this version of PE, nodes that include MCollective or ActiveMQ profiles trigger a warning during agent runs. Future versions of PE that remove the profiles completely can trigger failures in catalog compilation if you leave the profiles in place.

Test modules before upgrade

To ensure that your modules work with the newest version of PE, update and test them with Puppet Development Kit (PDK) before upgrading.

Before you begin

If you are already using PDK, your modules should pass validation and unit tests with your currently installed version of PDK.

Update PDK with each new release to ensure compatibility with new versions of PE.

1. Download and install PDK. If you already have PDK installed, this updates PDK to its latest version. For detailed instructions and download links, see the installing instructions.

2. If you have not previously used PDK with your modules, convert them to a PDK compatible format. This makes changes to your module to enable validation and unit testing with PDK. For important usage details, see the converting modules documentation.

For example, from within the module directory, run:

pdk convert
3. If your modules are already compatible with PDK, update them to the latest module template. If you converted modules in step 2, you do not need to update the template. To learn more about updating, see the updating module templates documentation.

For example, from within the module directory, run:

```
pdk update
```

4. Validate and run unit tests for each module, specifying the version of PE you are upgrading to. When specifying a PE version, be sure to specify at least the year and the release number, such as 2018.1. For information about module validations and testing, see the validating and testing modules documentation.

For example, from within the module directory, run:

```
pdk validate
pdk test unit
```

The `pdk test unit` command verifies that testing dependencies and directories are present and runs the unit tests that you write. It does not create unit tests for your module.

5. If your module fails validation or unit tests, make any necessary changes to your code.

After you've verified that your modules work with the new PE version, you can continue with your upgrade.

**Upgrade PE**

Upgrade PE infrastructure components to get the latest features and fixes. Follow the upgrade instructions for your installation type to ensure you upgrade components in the correct order. Coordinate upgrades to ensure all infrastructure nodes are upgraded in a timely manner, because agent runs and replication fail on infrastructure nodes running a different agent version than the primary server.

**Before you begin**

Review the upgrade cautions for major changes to architecture and infrastructure components which might affect your upgrade.

**Configure non-production environment for infrastructure nodes**

If your infrastructure nodes are in an environment other than production, you must manually configure PE to use your chosen environment before you upgrade.

**Important:** You only need to follow these steps if your infrastructure nodes are in an environment that is not production.

1. In `pe.conf`, set both of these parameters to the environment your infrastructure nodes are in:
   - `pe_install::install::classification::pe_node_group_environment`
   - `puppet_enterprise::master::recover_configuration::pe_environment`

2. Run `puppet-enterprise-installer -- --pe-environment <env>`

**Upgrade a standard installation**

To upgrade a standard installation, run the PE installer on your primary server, and then upgrade any additional components.

**Before you begin**

Back up your PE installation.

If you're upgrading a replica, ensure you have a valid admin RBAC token. If you're upgrading from 2018.1, the RBAC token must be generated by a user with Job orchestrator and Node group view permissions.
Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

1. If you're upgrading from a version earlier than 2019.7, the upgrade process includes a significant PuppetDB migration. You can expedite this process by deleting report from PuppetDB before upgrading. For instructions, refer to PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7 on page 259.

2. **Download** the tarball appropriate to your operating system and architecture.

3. Unpack the installation tarball: `tar -xf <tarball>`
   - You need about 1 GB of space to untar the installer.

4. From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:
   ```
   sudo ./puppet-enterprise-installer
   ```
   - If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`
     - With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

5. Upgrade these additional PE infrastructure components.
   - **Agents**
   - **PE client tools** — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

6. In disaster recovery installations, upgrade your replica.
   - The replica is temporarily unavailable to serve as backup during this step, so time upgrading your replica to minimize risk.
   a) On your primary server logged in as root, run: `sudo puppet infrastructure upgrade replica <REPLICA_FQDN>`
     - To specify the location of an authentication token other than the default: `sudo puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file <PATH_TO_TOKEN>`
   b) After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run: `sudo /opt/puppetlabs/bin/puppet-infra status`
   c) If your replica reports errors, reinitialize the replica. On your replica, run: `sudo /opt/puppetlabs/bin/puppet-infra reinitialize replica -y`

7. Optional: Remove old PE packages from all infrastructure nodes by running, from your primary server: `puppet infrastructure run remove_old_pe_packages`
   - All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>`

**Related information**

- **Back up your infrastructure** on page 842
- PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.
- **Generate a token using puppet-access** on page 349
Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

**Upgrade a large installation**

To upgrade a large installation, run the PE installer on your primary server, and then upgrade compilers and any additional components.

**Before you begin**

Back up your PE installation.

Ensure you have a valid admin RBAC token in order to upgrade compilers or a replica. If you're upgrading **from 2018.1**, the RBAC token must be generated by a user with **Job orchestrator** and **Node group** view permissions.

Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

1. If you're upgrading from a version earlier than 2019.7, the upgrade process includes a significant PuppetDB migration. You can expedite this process by deleting report from PuppetDB before upgrading. For instructions, refer to **PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7** on page 259.
2. Download the tarball appropriate to your operating system and architecture.
3. Unpack the installation tarball: `tar -xf <tarball>`
   You need about 1 GB of space to untar the installer.
4. From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:
   ```bash
   sudo ./puppet-enterprise-installer
   ```
   * If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`
     With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.
5. To upgrade compilers, on your primary server logged in as root, run: `puppet infrastructure upgrade compiler <COMPILER_FQDN-1>,<COMPILER_FQDN-2>`
   * To upgrade all compilers simultaneously: `sudo puppet infrastructure upgrade compiler --all`
   * To specify the location of an authentication token other than the default: `sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file <PATH_TO_TOKEN>`
6. Upgrade these additional PE infrastructure components.
   * Agents
   * PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.
7. In disaster recovery installations, upgrade your replica.

   The replica is temporarily unavailable to serve as backup during this step, so time upgrading your replica to minimize risk.

   a) On your primary server logged in as root, run:

   ```bash
   sudo puppet infrastructure upgrade replica <REPLICA_FQDN>
   ```

   To specify the location of an authentication token other than the default:

   ```bash
   sudo puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file <PATH_TO_TOKEN>
   ```

   b) After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run:

   ```bash
   sudo /opt/puppetlabs/bin/puppet-infra status
   ```

   c) If your replica reports errors, reinitialize the replica. On your replica, run:

   ```bash
   sudo /opt/puppetlabs/bin/puppet-infra reinitialize replica -y
   ```

8. Optional: Remove old PE packages from all infrastructure nodes by running, from your primary server:

   ```bash
   puppet infrastructure run remove_old_pe_packages
   ```

   All packages older than the current version are removed by default. To remove specific versions, append
   `pe_version=<VERSION_NUMBER>`

   Optionally convert legacy compilers to the new style compiler running the PuppetDB service.

   **Related information**

   Back up your infrastructure on page 842

   PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

   Generate a token using puppet-access on page 349

   Use puppet-access to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

   **Upgrade an extra-large installation**

   For help upgrading an extra-large installation, reach out to your technical account manager.

   **Upgrade a standalone PE-PostgreSQL installation**

   To upgrade a large installation with standalone PE-PostgreSQL, run the PE installer first on your PE-PostgreSQL node, then on your primary server, and then upgrade any additional components.

   **Before you begin**

   Back up your PE installation.

   Ensure you have a valid admin RBAC token in order to upgrade compilers. If you're upgrading from 2018.1, the RBAC token must be generated by a user with Job orchestrator and Node group view permissions.

   Remove from the console (in the PE Master node group), Hiera, or pe.conf any agent_version parameters that you've set in the pe_repo class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

   1. If you're upgrading from a version earlier than 2019.7, the upgrade process includes a significant PuppetDB migration. You can expedite this process by deleting report from PuppetDB before upgrading. For instructions, refer to PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7 on page 259.

   2. Download the tarball appropriate to your operating system and architecture.

   3. Unpack the installation tarball:

   ```bash
   tar -xf <tarball>
   ```

   You need about 1 GB of space to untar the installer.
4. Upgrade your PostgreSQL node.
   a) Ensure that the `pe.conf` file on your PostgreSQL node is up to date by running `puppet infrastructure recover_configuration` on your primary server, and then copying `/etc/puppetlabs/enterprise/conf.d` to the PostgreSQL node.
   b) Copy the installation tarball to the PostgreSQL node, and from the installer directory, run the installer:
      ```
      sudo ./puppet-enterprise-installer
      ```

5. From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:
   ```
   sudo ./puppet-enterprise-installer
   ```
   - If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag:
     ```
     sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
     ```
     With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

6. To upgrade compilers, on your primary server logged in as root, run: `puppet infrastructure upgrade compiler <COMPILER_FQDN-1>,<COMPILER_FQDN-2>`
   - To upgrade all compilers simultaneously: `sudo puppet infrastructure upgrade compiler --all`
   - To specify the location of an authentication token other than the default: `sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file <PATH_TO_TOKEN>`

7. Upgrade these additional PE infrastructure components.
   - Agents
   - PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

8. Optional: Remove old PE packages from all infrastructure nodes by running, from your primary server: `puppet infrastructure run remove_old_pe_packages`
   - All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>`
   Optionally convert legacy compilers to the new style compiler running the PuppetDB service.

Related information

- **Back up your infrastructure** on page 842
PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

- **Generate a token using puppet-access** on page 349
Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

**Upgrade an unmanaged PostgreSQL installation**

To upgrade a PE installation that relies on an unmanaged PostgreSQL database, you must first upgrade PostgreSQL to version 11, if necessary. Then, prepare your Puppet Enterprise (PE) databases and `pe.conf` file, and finally upgrade PE. Upgrade steps vary slightly depending on whether you use password or SSL authentication.

**Before you begin**

- Back up your PE installation.
- Ensure you have a valid admin RBAC token in order to upgrade compilers.
Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

**Important:** These steps use the default file and directory locations, and these steps assume the PostgreSQL superuser is `postgres`. Change these to match your installation as needed.

1. If you are upgrading from a PE version earlier than 2019.2.0, upgrade PostgreSQL.
   a) On your PostgreSQL node, install these packages:
      - `postgresql11`
      - `postgresql11-libs`
      - `postgresql11-contrib`
      - `postgresql11-server`
   
   b) Stop the PostgreSQL 9.6 service:
      ```bash
      systemctl stop postgresql-9.6.service
      ```
   
   c) As the PostgreSQL user, create a new database:
      ```bash
      su - postgres
      /usr/pgsql-11/bin/pg_ctl -D /var/lib/pgsql/11/data/ initdb
      ```
   
   d) If you use certificate authentication, take note of the location of your certificate, private key, and certificate authority files. You may need to move them to the new install directory, and you may need to update `postgresql.conf`.
   
   e) Copy your `postgresql.conf` file from `/var/lib/pgsql/9.6/data/` to `/var/lib/pgsql/11/data/` and update the `data` directory setting. If you moved the certificate, private key, and certificate authority files, update the `ssl_cert_file`, `ssl_key_file`, and `ssl_ca_file` settings.
   
   f) Migrate databases to PostgreSQL 11:
      ```bash
      ```
   
   g) Copy your `pg_hba.conf` and `pg_ident.conf` files from `/var/lib/pgsql/9.6/data/` to `/var/lib/pgsql/11/data/`. 
2. If necessary, update **pg_hba.conf** to ensure all relevant PE database roles have access.

This is an example of how this would look when using certificate authentication:

```plaintext
hostssl pe-activity pe-activity 0.0.0.0/0 cert map=pe-activity-pe-activity-map clientcert=1
hostssl pe-activity pe-activity-read 0.0.0.0/0 cert map=pe-activity-pe-activity-read-map clientcert=1
hostssl pe-activity pe-activity-write 0.0.0.0/0 cert map=pe-activity-pe-activity-write-map clientcert=1
hostssl pe-classifier pe-classifier 0.0.0.0/0 cert map=pe-classifier-pe-classifier-map clientcert=1
hostssl pe-classifier pe-classifier-read 0.0.0.0/0 cert map=pe-classifier-pe-classifier-read-map clientcert=1
hostssl pe-classifier pe-classifier-write 0.0.0.0/0 cert map=pe-classifier-pe-classifier-write-map clientcert=1
hostssl pe-inventory pe-inventory 0.0.0.0/0 cert map=pe-inventory-pe-inventory-map clientcert=1
hostssl pe-inventory pe-inventory-read 0.0.0.0/0 cert map=pe-inventory-pe-inventory-read-map clientcert=1
hostssl pe-inventory pe-inventory-write 0.0.0.0/0 cert map=pe-inventory-pe-inventory-write-map clientcert=1
hostssl pe-orchestrator pe-orchestrator 0.0.0.0/0 cert map=pe-orchestrator-pe-orchestrator-map clientcert=1
hostssl pe-orchestrator pe-orchestrator-read 0.0.0.0/0 cert map=pe-orchestrator-pe-orchestrator-read-map clientcert=1
hostssl pe-orchestrator pe-orchestrator-write 0.0.0.0/0 cert map=pe-orchestrator-pe-orchestrator-write-map clientcert=1
hostssl pe-puppetdb pe-puppetdb 0.0.0.0/0 cert map=pe-puppetdb-pe-puppetdb-map clientcert=1
hostssl pe-puppetdb pe-puppetdb-migrator 0.0.0.0/0 cert map=pe-puppetdb-pe-puppetdb-migrator-map clientcert=1
hostssl pe-puppetdb pe-puppetdb-read 0.0.0.0/0 cert map=pe-puppetdb-pe-puppetdb-read-map clientcert=1
hostssl pe-rbac pe-rbac 0.0.0.0/0 cert map=pe-rbac-pe-rbac-map clientcert=1
hostssl pe-rbac pe-rbac-read 0.0.0.0/0 cert map=pe-rbac-pe-rbac-read-map clientcert=1
hostssl pe-rbac pe-rbac-write 0.0.0.0/0 cert map=pe-rbac-pe-rbac-write-map clientcert=1
local all postgres peer
```
3. If necessary, and you are using certificate authentication, update `pg_ident.conf` to ensure all relevant PE database roles are mapped appropriately.
   
   For example:

   ```
   pe-activity-pe-activity-map <PRIMARY_CERTNAME> pe-activity
   pe-activity-pe-activity-read-map <PRIMARY_CERTNAME> pe-activity-read
   pe-activity-pe-activity-write-map <PRIMARY_CERTNAME> pe-activity-write
   pe-classifier-pe-classifier-map <PRIMARY_CERTNAME> pe-classifier
   pe-classifier-pe-classifier-read-map <PRIMARY_CERTNAME> pe-classifier-read
   pe-classifier-pe-classifier-write-map <PRIMARY_CERTNAME> pe-classifier-write
   pe-inventory-pe-inventory-map <PRIMARY_CERTNAME> pe-inventory
   pe-inventory-pe-inventory-read-map <PRIMARY_CERTNAME> pe-inventory-read
   pe-inventory-pe-inventory-write-map <PRIMARY_CERTNAME> pe-inventory-write
   pe-orchestrator-pe-orchestrator-map <PRIMARY_CERTNAME> pe-orchestrator
   pe-orchestrator-pe-orchestrator-read-map <PRIMARY_CERTNAME> pe-orchestrator-read
   pe-orchestrator-pe-orchestrator-write-map <PRIMARY_CERTNAME> pe-orchestrator-write
   pe-puppetdb-pe-puppetdb-map <PRIMARY_CERTNAME> pe-puppetdb
   pe-puppetdb-pe-puppetdb-migrator-map <PRIMARY_CERTNAME> pe-puppetdb-migrator
   pe-puppetdb-pe-puppetdb-read-map <PRIMARY_CERTNAME> pe-puppetdb-read
   pe-rbac-pe-rbac-map <PRIMARY_CERTNAME> pe-rbac
   pe-rbac-pe-rbac-read-map <PRIMARY_CERTNAME> pe-rbac-read
   pe-rbac-pe-rbac-write-map <PRIMARY_CERTNAME> pe-rbac-write
   ```

4. Start or restart the PostgreSQL 11 service. If you migrated to 11, start the PostgreSQL 11 service for the first time. If you were already on 11 and you made changes to the `conf` files, restart the PostgreSQL 11 service.

   ```
   systemctl start postgresql-11
   ```
5. Prepare PE databases for upgrade.

a) As the PostgreSQL user, clean up the database:

```
su - postgres
/usr/pgsql-11/bin/vacuumdb --analyze --all
```

b) If you are upgrading from 2018.1 or 2019.0, create a directory for the pe-inventory tablespace. Depending on the version you are upgrading from, this step may not be necessary.

First, run:

```
mkdir -p <PATH TO TABLESPACES DIR>/pe-inventory
chown postgres:postgres <PATH TO TABLESPACES DIR>/pe-inventory
chmod 700 <PATH TO TABLESPACES DIR>/pe-inventory
```

Then, create the database by running the following commands at the psql prompt:

```
CREATE TABLESPACE "pe-inventory" LOCATION '
<path to tablespaces dir>/pe-inventory";
CREATE DATABASE "pe-inventory" OWNER "postgres" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0 TABLESPACE 'pe-inventory';
REVOKE CONNECT ON DATABASE "pe-inventory" FROM public;
```

c) Create any new or missing roles as necessary.

These commands include all roles. You may not need to create the base roles; however, if you are upgrading from 2018.1, you need to create the pe-puppetdb-migrator role as well as all -read and -write roles.

**Note:** The follow commands include passwords for the roles. You can omit passwords if you are using certificate authentication.

```
CREATE ROLE "pe-inventory" LOGIN NOCREATEROLE NOCREATEDB SUPERUSER CONNECTION LIMIT -1 PASSWORD '<INVENTORY_DATABASE_PASSWORD>';  
CREATE ROLE "pe-inventory-read" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<INVENTORY_DATABASE_PASSWORD>';  
CREATE ROLE "pe-inventory-write" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<INVENTORY_DATABASE_PASSWORD>';  
CREATE ROLE "pe-puppetdb" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<PUPPETDB_PASSWORD>';  
CREATE ROLE "pe-puppetdb-migrator" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<PUPPETDB_PASSWORD>';  
CREATE ROLE "pe-puppetdb-read" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<PUPPETDB_PASSWORD>';  
CREATE ROLE "pe-puppetdb-write" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<PUPPETDB_PASSWORD>';  
CREATE ROLE "pe-classifier" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<CLASSIFIER_DATABASE_PASSWORD>';  
CREATE ROLE "pe-classifier-read" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<CLASSIFIER_DATABASE_PASSWORD>';  
CREATE ROLE "pe-classifier-write" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<CLASSIFIER_DATABASE_PASSWORD>';  
CREATE ROLE "pe-rbac" LOGIN NOCREATEROLE NOCREATEDB SUPERUSER CONNECTION LIMIT -1 PASSWORD '<RBAC_DATABASE_PASSWORD>';  
CREATE ROLE "pe-rbac-read" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<RBAC_DATABASE_PASSWORD>';  
CREATE ROLE "pe-rbac-write" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<RBAC_DATABASE_PASSWORD>';  
CREATE ROLE "pe-activity" LOGIN NOCREATEROLE NOCREATEDB SUPERUSER CONNECTION LIMIT -1 PASSWORD '<ACTIVITY_DATABASE_PASSWORD>';  
CREATE ROLE "pe-activity-read" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<ACTIVITY_DATABASE_PASSWORD>';  
CREATE ROLE "pe-activity-write" LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD '<ACTIVITY_DATABASE_PASSWORD>';  
```

**© 2022 Puppet, Inc., a Perforce company**
6. Prepare your `pe.conf` file for upgrade.

**Password authentication**

Add your inventory database password and verify existing host, port, and password parameters.

```
"puppet_enterprise::puppet_master_host": "<PRIMARY_CERTNAME>"
"puppet_enterprise::database_host": "<DATABASE_NODE_CERTNAME>"
"puppet_enterprise::database_port": 5432
"puppet_enterprise::puppetdb_database_password": "<PUPPETDB_PASSWORD>"
"puppet_enterprise::classifier_database_password": "<CLASSIFIER_DATABASE_PASSWORD>"
"puppet_enterprise::activity_database_password": "<ACTIVITY_DATABASE_PASSWORD>"
"puppet_enterprise::rbac_database_password": "<RBAC_DATABASE_PASSWORD>"
"puppet_enterprise::orchestrator_database_password": "<ORCHESTRATOR_DATABASE_PASSWORD>"
"puppet_enterprise::inventory_database_password": "<INVENTORY_DATABASE_PASSWORD>
```

**Certificate authentication**

Verify existing port and SSL parameters, and remove any password parameters.

```
"puppet_enterprise::database_ssl": true
"puppet_enterprise::database_cert_auth": true
"puppet_enterprise::database_port": 5432
"puppet_enterprise::profile::console::database_properties": "?ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=/etc/puppetlabs/puppet/ssl/certs/<PRIMARY_CERTNAME>.cert.pem&sslkey=/etc/puppetlabs/puppet/ssl/certs/<PRIMARY_CERTNAME>.private_key.pk8&sslcert=/etc/puppetlabs/puppetdb/ssl/<PRIMARY_CERTNAME>.cert.pem"
"puppet_enterprise::profile::puppetdb::database_properties": "?ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=/etc/puppetlabs/puppetdb/ssl/<PRIMARY_CERTNAME>.cert.pem&sslkey=/etc/puppetlabs/puppetdb/ssl/<PRIMARY_CERTNAME>.private_key.pk8&sslcert=/etc/puppetlabs/puppetdb/ssl/<PRIMARY_CERTNAME>.cert.pem"
"puppet_enterprise::profile::orchestrator::database_properties": "?ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=/etc/puppetlabs/orchestration-services/ssl/<PRIMARY_CERTNAME>.cert.pem&sslkey=/etc/puppetlabs/orchestration-services/ssl/<PRIMARY_CERTNAME>.private_key.pk8&sslcert=/etc/puppetlabs/orchestration-services/ssl/<PRIMARY_CERTNAME>.cert.pem"
```

7. If you're upgrading from a version earlier than 2019.7, the upgrade process includes a significant PuppetDB migration. You can expedite this process by deleting report from PuppetDB before upgrading. For instructions, refer to [PuppetDB migrations in PE 2019.1, 2019.3, and 2019.7](#) on page 259.

8. Download the tarball appropriate to your operating system and architecture.

9. Unpack the installation tarball: `tar -xf <tarball>`

   You need about 1 GB of space to untar the installer.

10. From the installer directory on your primary server, run the installer, using the `-c` flag to specify your existing `pe.conf` file and the `-s` flag to skip database checks by the installer. Follow the CLI instructions to complete your server upgrade.

   ```
   sudo ./puppet-enterprise-installer -s -c <FULL PATH TO pe.conf>
   ```
11. To upgrade compilers, on your primary server logged in as root, run: puppet infrastructure upgrade compiler <COMPILER_FQDN-1>,<COMPILER_FQDN-2>
   • To upgrade all compilers simultaneously: sudo puppet infrastructure upgrade compiler --all
   • To specify the location of an authentication token other than the default: sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file <PATH_TO_TOKEN>

12. Upgrade these additional PE infrastructure components.
   • Agents
     • PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

13. Optional: Remove old PE packages from all infrastructure nodes by running, from your primary server: puppet infrastructure run remove_old_pe_packages
    All packages older than the current version are removed by default. To remove specific versions, append pe_version=<VERSION_NUMBER>

    Optionally convert legacy compilers to the new style compiler running the PuppetDB service.

Migrate PE

As an alternative to upgrading, you can migrate your PE installation. Migrating results in little or no downtime, but it requires additional system resources because you must configure a new primary server.

**Note:** This procedure is not supported for installations with disaster recovery.

**Migrate a standard installation**

Migrate a standard installation by standing up a new primary server, restoring it with your existing installation, upgrading it, and then pointing agents at the new primary.

**Before you begin**

Review the upgrade cautions for major changes to architecture and infrastructure components which might affect your upgrade.

1. **Install** your current PE version on a new primary server node.
2. **Back up** your existing installation.
3. **Restore** your installation on the new primary server using the backup you created.
4. **Upgrade** your new primary server to the latest PE version.
5. Update nodes' puppet.conf server setting to the hostname for your new primary server:

   ```
   sudo puppet config set server <PRIMARY_HOSTNAME>
   ```

**Upgrading agents**

Upgrade your agents as new versions of Puppet Enterprise become available. The puppet_agent module helps automate upgrades, and provides the safest upgrade. Alternatively, you can upgrade individual nodes using a script.

**Note:** Before upgrading agents, verify that the primary server and agent software versions are compatible. Then after upgrade, run Puppet on your agents as soon as possible to verify that agents have the correct configuration and that your systems are behaving as expected.
Setting your desired agent version

To upgrade your primary server but use an older agent version that is still compatible with the new primary server, define a `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>` class with the `agent_version` variable set to your desired agent version.

To ensure your agents are always running the same version as your primary server, in the puppetlabs-puppet_agent module, set the `package_version` variable for the `puppet_agent` class to `auto`. This causes agents to automatically upgrade themselves on their first Puppet run after a primary server upgrade.

**Related information**

Upgrading Puppet Enterprise on page 257
Upgrade your PE installation as new versions become available.

Upgrade agents using the puppet_agent module

The puppetlabs-puppet_agent module, available from the Forge, enables you to upgrade multiple *nixor Windows agents at one time. The module handles all the latest version-to-version upgrades.

**Important:** For the most reliable upgrade, always use the latest version of the puppet_agent module to upgrade agents. Test the upgrade on a subset of agents, and after you verify the upgrade, upgrade remaining agents.

1. On your primary server, download and install the puppetlabs-puppet_agent module: `puppet module install puppetlabs-puppet_agent`

2. Configure the primary server to download the agent version you want to upgrade.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
   b) On the Classes tab, in the Add a new class field, enter `pe_repo`, and select the appropriate repo class from the list of classes.
      Repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.
      To specify a particular agent version, set the `agent_version` variable using an X.Y.Z format (for example, 5.5.14). When their version is set explicitly, agents do not automatically upgrade when you upgrade your primary server.
   c) Click Add class and commit changes.
   d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`
      The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.

3. Click Node groups, click Add group, specify options for a new upgrade node group, and then click Add.
   • Parent name — Select the name of the classification node group that you want to set as the parent to this group, in this case, All Nodes.
   • Group name — Enter a name that describes the role of this classification node group, for example, agent_upgrade.
   • Environment — Select the environment your agents are in.
   • Environment group — Do not select this option.

4. Click the link to Add membership rules, classes, and variables.

5. On the Rules tab, create a rule to add the agents that you want to upgrade to this group, click Add Rule, and then commit changes.
   For example:
   • Fact — osfamily
   • Operator — =
   • Value — RedHat

© 2022 Puppet, Inc., a Perforce company
6. Still in the agent upgrade group, on the Classes tab, add the puppet_agent class, and click Add class.
   If you don’t immediately see the class, click Refresh to update the classifier.

   **Note:** If you’ve changed the prefix parameter of the pe_repo class in your PE Master node group, set the puppet-agent source parameter of the upgrade group to `https://<PRIMARY_HOSTNAME>:8140/<Prefix>`.

7. In the puppet_agent class, specify the version of the puppet-agent package version that you want to install, then commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_version</td>
<td>The puppet-agent package version to install, for example 5.3.3.</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to auto to install the same agent version that is installed on your primary server.</td>
</tr>
</tbody>
</table>

8. On the agents that you’re upgrading, run Puppet: `/opt/puppet/bin/puppet agent -t`

   After the Puppet run, you can verify the upgrade with `/opt/puppetlabs/bin/puppet --version`

**Upgrade agents using a script**

To upgrade an individual node, for example to test or troubleshoot, you can upgrade directly from the node using a script. This method relies on a package repository hosted on your primary server.

   **Note:** If you encounter SSL errors during the upgrade process, ensure your agent's OpenSSL is up to date and matches the primary server's version. You can check the primary server's OpenSSL versions with `/opt/puppetlabs/puppet/bin/openssl version` and the agent's version with `openssl version`.

**Upgrade a *nix agent using a script**

You can upgrade an individual *nix agent using a script.

1. Configure the primary server to download the agent version you want to upgrade.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
   b) On the Classes tab, in the Add a new class field, enter pe_repo, and select the appropriate repo class from the list of classes.

   Repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

   To specify a particular agent version, set the agent_version variable using an `X.Y.Z` format (for example, 5.5.14). When their version is set explicitly, agents do not automatically upgrade when you upgrade your primary server.
   c) Click Add class and commit changes.
   d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`

     The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.

2. SSH into the agent node you want to upgrade.
3. Run the upgrade command appropriate to the operating system.

   - Most *nix
     
     ```bash
     cacert="$(puppet config print localcacert)"
     url="https://$(puppet config print server):8140/packages/current/install.bash"
     
     curl --cacert "$cacert" "$uri" | sudo bash
     ```
     
     See Usage notes for curl examples for information about forming curl commands.

   - Mac OS X, Solaris, and AIX
     
     ```bash
     cacert="$(puppet config print localcacert)"
     url="https://$(puppet config print server):8140/packages/current/install.bash"
     
     curl --cacert "$cacert" "$uri" | sudo bash
     ```

PE services restarts automatically after upgrade.

**Upgrade a Windows agent using a script**

You can upgrade an individual Windows agent using a script. For Windows, this method is riskier than using the puppet_agent module to upgrade, because you must manually complete and verify steps that the module handles automatically.

**Note:** The `<PRIMARY_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. Stop the Puppet service and the PXP agent service.
2. On the Windows agent, run the install script:

   ```powershell
   ```

3. Verify that Puppet runs are complete.
4. Restart the Puppet service and the PXP agent service.

**Upgrade agents without internet access**

In situations where your primary and agents are airgapped, the primary server can't download the package. Therefore, you have to download the agent tarball from an internet-connected system, prepare the airgapped primary server to serve up the agent package to your agents, and then run the upgrade script on your agents.

1. **Download** the appropriate agent tarball.

   If you are installing an agent version that is different from your primary server, make sure you download the agent tarball corresponding to the `agent_version` parameter for the node's platform, as explained in Setting agent versions on page 277.

   ```powershell
   2. On your primary server, copy the agent tarball to the appropriate agent package directory at: /opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-<AGENT_VERSION>

© 2022 Puppet, Inc., a Perforce company
3. Declare the agent architecture class in the **PE Master** node group:
   a) In the PE console, go to **Node groups > PE Infrastructure > PE Master**.
   b) On the **Classes** tab, enter `pe_repo` in the **Add a new class** field, and select the appropriate repo class from the list of classes.
   
   Repo classes are formatted as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.
   
   To use a specific agent version, set the `agent_version` variable using an X.Y.Z format (for example, 5.5.14). If you specify a version in this way, agents **do not** automatically upgrade when you upgrade your primary server.
   c) Click **Add class** and commit changes.
   d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`

   A new agent package repo is created at `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.

4. SSH into, or physically log on to, the agent node you want to upgrade.

5. Run the upgrade command appropriate to the operating system.
   - Most *nix
     
     ```
     cacert="$(puppet config print localcacert)"
     url="https://$(puppet config print server):8140/packages/current/install.bash"
     
     curl --cacert "$cacert" "$uri" | sudo bash
     ```

     See Usage notes for curl examples for information about forming curl commands.
   - Mac OS X, Solaris, and AIX
     
     ```
     cacert="$(puppet config print localcacert)"
     url="https://$(puppet config print server):8140/packages/current/install.bash"
     
     curl --cacert "$cacert" "$uri" | sudo bash
     ```

6. Repeat these steps to upgrade additional agents.

### Setting agent versions

Usually, you want your agent nodes to run the same agent version as the primary server; however, if absolutely necessary, agent nodes can run a different, but compatible, version.

**Important:** Make sure the primary server and agent version are compatible. Component versions in recent PE releases on page 14 lists which Puppet agent version are tested and supported for each PE release.

If you Upgrade agents using the puppet_agent module on page 274, you specify the agent version by setting the `package_version` parameter on the agent upgrade node group. You can define a specific version or set this to `auto`, if you want your agents to always run the same version as your primary server. When set to `auto`, agent nodes to automatically upgrade themselves on their first Puppet run after a primary server upgrade. You can also set the `package_version` parameter for the `puppet_agent` class in the `puppet_agent` module's configuration.

The agent version can be specified on a platform-by-platform basis by the `agent_version` parameter of any `pe_repo::platform` classes in the **PE Master** node group (at **Node Groups > PE Master > Classes**). If your nodes run on various platforms, you must set the `agent_version` on each `pe_repo` class that you want to use a specific agent version. For example, you can specify different versions for 32-bit and 64-bit Windows agents.

**CAUTION:**

Setting `agent_version` blocks upgrades. Setting this parameter is only recommended in specific scenarios with strong justification for doing so.
Never set `agent_version` for infrastructure nodes. Critical failures can occur if all your infrastructure nodes, including the primary server, compilers, and replicas, aren't running the same agent version.

When you install or upgrade agent nodes, the agent install script looks at the node's platform class and installs the specified agent version. If you don't specify a version for a platform, the script installs the default version packaged with your current version of PE. If you specified an older version for your agent platforms, you could upgrade your primary server while maintaining an older agent version on your agent nodes. Similarly, if you specified a newer version for your agent platforms, your agent nodes would run a newer agent version than your primary server.

**CAUTION:**

The primary server's agent version must match the agent version on other infrastructure nodes, including compilers and replicas, otherwise your primary server won’t compile catalogs for those nodes. Not compiling catalogs is a critical failure. Never set `agent_version` on any infrastructure node (including the primary server, compilers, and replicas).

**Related information**

Component versions in recent PE releases on page 14

These tables show which components are in Puppet Enterprise (PE) 2019.y and 2018.y. releases. Component version tables for prior releases are available in the [Documentation for other PE versions](#) on page 30. Component version tables for the current STS and LTS releases are available in the [latest PE documentation](#).
Configuring Puppet Enterprise

- **Tuning infrastructure nodes** on page 279
  Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.
- **Methods for configuring PE** on page 286
  After you've installed Puppet Enterprise (PE), optimize it for your environment by configuring and tuning settings as needed. For example, you might want to add your certificate to the allowlist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.
- **Configuring Puppet Server** on page 288
  After you've installed Puppet Enterprise, optimize it for your environment by configuring Puppet Server settings as needed.
- **Configuring PuppetDB** on page 292
  After you've installed Puppet Enterprise, optimize it for your environment by configuring PuppetDB as needed.
- **Configuring security settings** on page 294
  Ensure your PE environment is secure by configuring security settings.
- **Configuring proxies** on page 298
  You can work around limited internet access by configuring proxies at various points in your infrastructure, depending on your connectivity limitations.
- **Configuring the console** on page 299
  After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.
- **Configuring orchestration** on page 302
  After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.
- **Configuring ulimit** on page 306
  As your infrastructure grows and you bring more agents under management, you might need to increase the number allowed file handles per client.
- **Writing configuration files** on page 307
  Puppet supports two formats for configuration files that configure settings: valid JSON and Human-Optimized Config Object Notation (HOCON), a JSON superset.
- **Analytics data collection** on page 308
  Some components automatically collect data about how you use Puppet Enterprise. If you want to opt out of providing this data, you can do so, either during or after installing.
- **Static catalogs** on page 312
  A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing `source` attributes pointing to `puppet://` locations on a node.

**Tuning infrastructure nodes**

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

PE is composed of multiple services on one or more infrastructure hosts. Each service has multiple settings that can be configured to maximize use of system resources and optimize performance. The default settings for each service are conservative, because the set of services sharing resources on each host varies depending on your infrastructure. Optimized settings vary depending on the complexity and scale of your infrastructure.

Configure settings after an install or upgrade, or after making changes to infrastructure hosts, including changing the system resources of existing hosts, or adding new hosts, including compilers.

© 2022 Puppet, Inc., a Perforce company
**Related information**

**Hardware requirements** on page 182
These hardware requirements are based on internal testing at Puppet and are provided as minimum guidelines to help you determine your hardware needs.

**Primary server tuning**
These are the default and recommended tuning settings for your primary server or disaster recovery replica.

**Note:** Recommended settings are appropriate for standard installations or large installations with compilers running the PuppetDB service. Installations with legacy compilers generally require more resources on the primary server for PuppetDB.

<table>
<thead>
<tr>
<th></th>
<th>Puppet Server</th>
<th>PuppetDB</th>
<th>Console</th>
<th>Orchestator</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JRuby max active instances</td>
<td>Java heap (MB)</td>
<td>ReservedCommand processing threads</td>
<td>Java heap (MB)</td>
<td>Java heap (MB)</td>
</tr>
<tr>
<td>4 cores, 8 GB RAM</td>
<td>Default 3</td>
<td>2048</td>
<td>512</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Recommended</td>
<td>1024</td>
<td>192</td>
<td>1</td>
<td>819</td>
</tr>
<tr>
<td></td>
<td>With legacy compilers</td>
<td>2</td>
<td>1024</td>
<td>192</td>
<td>2</td>
</tr>
<tr>
<td>6 cores, 10 GB RAM</td>
<td>Default 4</td>
<td>2048</td>
<td>512</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Recommended</td>
<td>2304</td>
<td>288</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>With legacy compilers</td>
<td>2</td>
<td>1536</td>
<td>192</td>
<td>3</td>
</tr>
<tr>
<td>8 cores, 12 GB RAM</td>
<td>Default 4</td>
<td>2048</td>
<td>512</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Recommended</td>
<td>2304</td>
<td>288</td>
<td>2</td>
<td>1228</td>
</tr>
<tr>
<td></td>
<td>With legacy compilers</td>
<td>3</td>
<td>2304</td>
<td>288</td>
<td>4</td>
</tr>
<tr>
<td>10 cores, 16 GB RAM</td>
<td>Default 4</td>
<td>2048</td>
<td>512</td>
<td>5</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Recommended</td>
<td>3840</td>
<td>480</td>
<td>2</td>
<td>1638</td>
</tr>
<tr>
<td></td>
<td>With legacy compilers</td>
<td>4</td>
<td>3072</td>
<td>384</td>
<td>5</td>
</tr>
<tr>
<td>12 cores, 24 GB RAM</td>
<td>Default 4</td>
<td>2048</td>
<td>512</td>
<td>6</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Recommended</td>
<td>6144</td>
<td>768</td>
<td>3</td>
<td>2457</td>
</tr>
<tr>
<td></td>
<td>With legacy compilers</td>
<td>5</td>
<td>3840</td>
<td>480</td>
<td>6</td>
</tr>
</tbody>
</table>
Compiler tuning

These are the default and recommended tuning settings for compilers running the PuppetDB service.

<table>
<thead>
<tr>
<th>Puppet Server</th>
<th>PuppetDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRuby max active instances</td>
<td>Java heap (MB)</td>
</tr>
<tr>
<td>4 cores, 8 GB RAM</td>
<td>Default</td>
</tr>
<tr>
<td>6 cores, 12 GB RAM</td>
<td>Default</td>
</tr>
</tbody>
</table>

Legacy compiler tuning

These are the default and recommended tuning settings for legacy compilers without the PuppetDB service.

| Puppet Server |  |
|---------------|  |
| JRuby max active instances | Java heap (MB) | Reserved code cache (MB) |
| 4 cores, 8 GB RAM | Default | 3 | 2048 | 512 |
| 6 cores, 12 GB RAM | Default | 4 | 2048 | 512 |

Using the puppet infrastructure tune command

The puppet infrastructure tune command outputs optimized settings for PE services based on recommended guidelines.

When you run puppet infrastructure tune, it queries PuppetDB to identify infrastructure hosts and their processor and memory facts, and outputs settings in YAML format for use in Hiera.

The command is compatible with most standard PE configurations, including those with compilers, a replica, or standalone PE-PostgreSQL.

The command must be run on your primary server as root. Running with elevated privileges via sudo puppet infrastructure is not sufficient. Instead, start a root session by running sudo su -, and then run the puppet infrastructure command.

These are the options commonly used with the puppet infrastructure tune command:

- `--current` outputs existing tuning settings from the console and Hiera, and identifies duplicate settings found in both places.
- `--memory_per_jruby <MB>` outputs tuning recommendations based on specified memory allocated to each JRuby in Puppet Server. If you implement tuning recommendations using this option, specify the same value for puppetserver_ram_per_jruby.
- `--memory_reserved_for_os <MB>` outputs tuning recommendations based on specified RAM reserved for the operating system.
- `--common` outputs common settings — identical on several nodes — separately from node-specific settings.
For more information about the tune command, run `puppet infrastructure tune --help`.

**Related information**

**RAM per JRuby** on page 282

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server. In installations with compilers running the PuppetDB service, this setting is a good starting point for tuning your installation, because the value you specify is factored into several other parameters, including JRuby max active instances and heap allocation on compilers running PuppetDB.

**Tuning parameters**

Tuning parameters let you customize PE components for maximum performance and hardware resource utilization.

Specify tuning parameters using Hiera for the best scalability and consistency. If you must use the console, add the parameter to the appropriate infrastructure node group using the method suitable for the parameter type:

- Specify `puppet_enterprise::profile` parameters, including `java_args`, `shared_buffers`, and `work_mem`, as parameters of their class.
- Specify all other tuning parameters as configuration data.

**RAM per JRuby**

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server. In installations with compilers running the PuppetDB service, this setting is a good starting point for tuning your installation, because the value you specify is factored into several other parameters, including JRuby max active instances and heap allocation on compilers running PuppetDB.

**Parameter**

```
puppet_enterprise::puppetserver::puppetserver::ram_per_jruby
```

**Default value**

512 MB

**Accepted values**

- Integer (MB)

**How to calculate**

If you have complex Hiera code, many environments or modules, or large reports, you might need to increase this setting. You can generally achieve good performance by allocating up to around 2 GB per JRuby. If 2 GB is inadequate, you might benefit from enabling environment caching.

**Console node group**

PE Master

**JRuby max active instances**

The `jruby_max_active_instances` setting controls the maximum number of JRuby instances to allow on the Puppet Server, or how many plans can run concurrently in the orchestrator. Each plan uses one JRuby instance, and nested plans use their calling plan's JRuby.

**Parameter**

Puppet Server —

```
puppet_enterprise::master::puppetserver::jruby_max_active_instances
```

**Tip:** This setting is referred to as `max_active_instances` in the `pe-puppet-server.conf` file and in open source Puppet. It's the same setting.

Orchestration services —

```
puppet Enterprise::profile::orchestrator::jruby_max_active_instances
```

**Default value**

Primary server — Number of CPUs - 1, minimum 1, maximum 4
Compilers — Number of CPUs x 0.75, minimum 1, maximum 24
Orchestration services — Orchestrator heap size ($\text{java_args}/1024$, minimum 1

**Accepted values**
Integer

**How to calculate**
Puppet Server — As a conservative estimate, one JRuby process uses approximately 512 MB of RAM. Four JRuby instances works for most environments. Because increasing the maximum number of JRuby instances also increases the amount of RAM used by Puppet Server, make sure the Puppet Server heap size ($\text{java_args}$) is scaled proportionally. For example, if you set $\text{jruby_max_active_instances}$ to 4, set Puppet Server $\text{java_args}$ to at least 2 GB.

Orchestration services — Setting the orchestrator heap size ($\text{java_args}$) automatically sets the number of JRuby instances available inside orchestrator. For example, setting the orchestrator heap size to 5120 MB enables a maximum of five JRuby instances, or plans, to run concurrently. Enabling too many JRuby instances might reduce system performance, especially if the plans you're running use a lot of memory. Increase the orchestrator heap size if you notice poor performance while running plans.

**Note:** JRuby instances are used only by plans running in the orchestrator, and they are deallocated once a plan finishes running. Tasks are not affected by JRubys.

**Console node group**
Puppet Server — PE Master or, for compilers running the PuppetDB service, PE Compiler
Orchestration services — PE Orchestrator

**JRuby max requests per instance**
The $\text{jruby_max_requests_per_instance}$ setting determines the maximum number of HTTP requests a JRuby handles before it’s terminated. When a JRuby instance reaches this limit, it's flushed from memory and replaced with a fresh one.

**Parameter**
```
puppet_enterprise::master::puppetserver::jruby_max_requests_per_instance
```

**Tip:** This setting is referred to as $\text{max_requests_per_instance}$ in the `pe-puppet-server.conf` file and in open source Puppet. It's the same setting.

**Default value**
100,000

**Accepted values**
Integer

**How to calculate**
More frequent JRuby flushing can help address memory leaks, because it prevents any one interpreter from consuming too much RAM. However, performance is reduced slightly each time a new JRuby instance loads. Ideally, set this parameter to get a new interpreter no more than every few hours. There are multiple interpreters running with requests balanced across them, so the lifespan of each interpreter varies.

**Console node group**
PE Master
Java heap
The \texttt{java\_args} setting is used to specify \textit{heap size}: the amount of memory that each Java process is allowed to request from the operating system. You can specify heap size for each PE service that uses Java, including Puppet Server, PuppetDB, and console and orchestration services.

Heap size is specified as \texttt{Xmx} and \texttt{Xms}, the maximum and minimum heap size, respectively. Typically, the maximum and minimum are set to the same value so that heap size is fixed, for example \{ 'Xmx' => '2048m', 'Xms' => '2048m' \}.

Parameters

- \textbf{Puppet Server} — puppet\_enterprise::profile::master::java\_args
- \textbf{Tip:} This setting might be referred to as puppet\_enterprise::master::java\_args or puppet\_enterprise::master::puppetserver::java\_args. They are all the same thing: profile::master filters down to master, which filters down to master::puppetserver.
- \textbf{PuppetDB} — puppet\_enterprise::profile::puppetdb
- \textbf{Console services} — puppet\_enterprise::profile::console
- \textbf{Orchestration services} — puppet\_enterprise::profile::orchestrator

Default values

- \textbf{Puppet Server} — 2 GB
- \textbf{PuppetDB} — 256 MB
- \textbf{Console services} — 256 MB
- \textbf{Orchestration services} — 704 MB

Accepted values

- JSON string

Console node group

- \textbf{Puppet Server} — PE Master or PE Compiler
- \textbf{PuppetDB} — PE PuppetDB or, for compilers running the PuppetDB service, PE Compiler
- \textbf{Console services} — PE Console
- \textbf{Orchestration services} — PE Orchestrator

Puppet Server reserved code cache
The \texttt{reserved\_code\_cache} setting specifies the maximum space available to store the Puppet Server code cache during catalog compilation.

Parameter

- \texttt{puppet\_enterprise::master::puppetserver::reserved\_code\_cache}

Default value

- \textbf{Primary server} — If total RAM is less than 2 GB, the Java default is used. Otherwise, 512 MB.
- \textbf{Compilers} — Number of JRuby instances \( \times \) 128 MB, min 128 MB, max 2048 MB

Accepted values

- Integer (MB)

How to calculate

JRuby requires an estimated 128 MB of cache space per instance, so to determine the minimum amount of space needed: number of JRuby instances \( \times \) 128 MB

Console node group

- \textbf{PE Master} or, for compilers running the PuppetDB service, PE Compiler
PuppetDB command processing threads
The `command_processing_threads` setting specifies how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.

Parameter
- `puppet_enterprise::puppetdb::command_processing_threads`

Default value
- Primary server — Number of CPUs x 0.5, minimum 1
- Compilers — Number of CPUs x 0.25, minimum 1, maximum 3

Accepted values
- Integer

How to calculate
- If the PuppetDB queue is backing up and you have CPU cores to spare, increasing the number of threads can help process the backlog more rapidly. Avoid allocating all of your CPU cores for command processing, as doing so can starve other PuppetDB subsystems of resources and actually decrease throughput.

Console node group
- PE PuppetDB or, for compilers running the PuppetDB service, PE Compiler

PostgreSQL shared buffers
The `shared_buffers` setting specifies the amount of memory the PE-PostgreSQL server uses for shared memory buffers.

Parameter
- `puppet_enterprise::profile::database::shared_buffers`

Default value
- Available RAM x 0.25, minimum 32 MB, maximum 4096 MB

Accepted values
- Integer (MB)

How to calculate
- The default of 25 percent of available RAM is suitable for most installations, but you might see improved console performance by increasing `shared_buffers` up to 40 percent of available RAM.

Console node group
- PE Database

PostgreSQL working memory
The `work_mem` setting specifies the maximum amount of memory used for queries before writing to temporary files.

Parameter
- `puppet_enterprise::profile::database::work_mem`

Default value
- `(Available RAM / 1024 / 8) + 0.5`, minimum 4, maximum 16

Accepted values
- Integer (MB)

Console node group
- PE Database
Methods for configuring PE

After you've installed Puppet Enterprise (PE), optimize it for your environment by configuring and tuning settings as needed. For example, you might want to add your certificate to the allowlist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

PE shares configuration settings used in open source Puppet and documented in the Configuration Reference, however PE defaults for certain settings might differ from the Puppet defaults. Some examples of settings that have different PE defaults include disable18n, environment_timeout, always_retry_plugins, and the Puppet Server JRuby max-active-instances setting. To verify PE configuration defaults, check the puppet.conf file after installation.

There are three main methods for configuring PE: using the console, adding a key to Hiera, or editing pe.conf. Be consistent with the method you choose, unless the situation calls for you to use a specific method over the others.

Configure settings using the console

The console allows you to use a graphical interface to configure Puppet Enterprise (PE).

Changes in the console override your Hiera data and data in pe.conf. It is usually best to use the console when you want to:

- Change parameters in profile classes starting with puppet_enterprise::profile.
- Add any parameters in PE-managed configuration files.
- Set parameters that configure at runtime.

There are two ways to change settings in the console: setting configuration data and editing parameters.

Related information

Preconfigured node groups on page 472
Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

Set configuration data

Configuration data set in the console is used for automatic parameter lookup, the same way that Hiera data is used. Console configuration data takes precedence over Hiera data, but you can combine data from both sources to configure nodes.

Tip: In most cases, setting configuration data in Hiera is the more scalable and consistent method, but there are some cases where the console is preferable. Use the console to set configuration data if:

- You want to override Hiera data. Data set in the console overrides Hiera data when configured as recommended.
- You want to give someone access to set or change data and they don’t have the skill set to do it in Hiera.
- You simply prefer the console user interface.

Important: If your installation includes a disaster recovery replica, make sure you’ve correctly enabled data editing in the console for both your primary server and replica.

1. In the console, click Node groups, then find the node group that you want to add configuration data to and select it.
2. On the Configuration data tab, specify a Class and select a Parameter to add.

   You can select from existing classes and parameters in the node group's environment, or you can specify free-form values. Classes aren’t validated, but any class you specify must be present in the node’s catalog at runtime in order for the parameter value to be applied.

   When you select a parameter, the Value field is automatically populated with the inherited or default value.
3. Optional: Change the default parameter Value.
Related information
Enable data editing in the console on page 302
The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

Set parameters
Parameters are declared resource-style, which means they can be used to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

1. In the console, click **Node groups**, and then find the node group that you want to add a parameter to and select it.
2. On the **Classes** tab, select the class you want to modify and the **Parameter** to add.

   The **Parameter** drop-down list shows all of the parameters that are available for that class in the node group’s environment. When you select a parameter, the **Value** field is automatically populated with the inherited or default value.
3. (Optional) Change the default **Value**.

Configure settings with Hiera
Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

**Before you begin**
For more information on how to use Hiera, see the Hiera docs.

Changes to PE configuration in Hiera override configuration settings in `pe.conf`, but not those set in the console. It's best to use Hiera when you want to:

- Change parameters in non-profile classes.
- Set parameters that are static and version controlled.
- Configure for high availability.

To configure a setting using Hiera:

1. Open your default data file.
   The default location for Hiera data files is:
   - *nix: `/etc/puppetlabs/code/environments/<ENVIRONMENT>/data/common.yaml`
   - Windows: `%CommonAppData%\PuppetLabs\code\environments\<ENVIRONMENT>\data\common.yaml`

   If you customize the `hiera.yaml` configuration to change location for data files (the `datadir` setting) or the path of the common data file (in the `hierarchy` section), look for the default .yaml file in the customized location.

2. Add your new parameter to the file in editor.
   For example, to increase the number of seconds before a node is considered unresponsive from the default 3600 to 4000, add the following to your .yaml default file and insert your new parameter at the end.

   ```
   Puppet_enterprise::console_services::no_longer_reporting_cutoff: 4000
   ```

3. To compile changes, run `puppet agent -t`

Related information
Preconfigured node groups on page 472
Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

**Configure settings in pe.conf**

Puppet Enterprise (PE) configuration data includes any data set in /etc/puppetlabs/enterprise/conf.d/ but pe.conf is the file used for most configuration activities during installation.

PE configuration settings made in Hiera and the console always override settings made in pe.conf. Configure settings using pe.conf when you want to:

- Access settings during installation.
- Configure for high availability.

To configure settings using pe.conf:

1. Open the pe.conf file on your primary server:

   ```
   /etc/puppetlabs/enterprise/conf.d/pe.conf
   ```

2. Add the parameter and new value you want to set.

   For example, to change the proxy in your repo, add the following and change the parameter to your new proxy location.

   ```
   pe_repo::http_proxy_host: "proxy.example.vlan"
   ```

3. Run puppet agent -t

   **Note:** If PE services are stopped, run puppet infrastructure configure instead of puppet agent -t.

**Configuring Puppet Server**

After you’ve installed Puppet Enterprise, optimize it for your environment by configuring Puppet Server settings as needed.

**Tune the Ruby load path**

The ruby_load_path setting determines where Puppet Server finds components such as Puppet and Facter.

The default setting is located at $puppetserver_jruby_puppet_ruby_load_path = ['/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib'].

**Note:** If you change the libdir you must also change the vardir.

To change the path to a different array in pe.conf:

1. Add the following to your pe.conf file on your primary server and set your new load path parameter.

   ```
   puppet_enterprise::master::puppetserver::puppetserver_jruby_puppet_ruby_load_path
   ```

2. Run puppet agent -t

**Tune the multithreaded server setting**

The jruby_puppet_multithreaded setting allows you to enable multithreaded mode, which uses a single JRuby instance to process requests, like catalog compiles, concurrently. The setting defaults to false.

To enable or disable the setting using Hiera:
1. Add the following code to your default .yaml file and set the parameter to true (enable) or false (disable).
   For example:
   ```yaml
   puppet_enterprise::master::puppetserver::jruby_puppet_multithreaded: <true or false>
   ```

2. To compile the changes, run puppet agent -t

### Enable or disable cached data when updating classes

The optional `environment-class-cache-enabled` setting specifies whether cached data is used when updating classes in the console. When true, Puppet Server refreshes classes using file sync, improving performance.

The default value for `environment-class-cache-enabled` depends on whether you use Code Manager.

- With Code Manager, the default value is enabled (true). File sync clears the cache automatically in the background, so clearing the environment cache manually isn't required when using Code Manager.
- Without Code Manager, the default value is disabled (false).

**Note:** If you're not using Code Manager and opt to enable this setting, make sure your code deployment method — for example r10k — clears the environment cache when it completes. If you don't clear the environment cache, the Node Classifier doesn't receive new class information until Puppet Server is restarted.

To enable or disable the cache using Hiera:

1. Add the following code to your default .yaml file and set the parameter to the appropriate setting.
   ```yaml
   puppet_enterprise::master::puppetserver::jruby_environment_class_cache_enabled: <true OR false>
   ```

2. To compile the changes, run puppet agent -t

### Change the environment_timeout setting

The `environment_timeout` setting controls how long the primary server caches data it loads from an environment, determining how much time passes before changes to an environment's Puppet code are reflected in its environment.

In PE, the `environment_timeout` is set to 0. This lowers the performance of your primary server but makes it easy for new users to deploy updated Puppet code. Once your code deployment process is mature, change this setting to `unlimited`.

**Note:** When you install Code Manager and set the `code_manager_auto_configure` parameter to true, `environment_timeout` is updated to `unlimited`.

To change the `environment_timeout` setting using `pe.conf`:

1. Add the following to your `pe.conf` file on your primary server and specify either 0 or unlimited:
   ```
   puppet_enterprise::master::environment_timeout:<time>
   ```

2. Run puppet agent -t

For more information, see Environments limitations

### Add certificates to the puppet-admin certificate allowlist

Add trusted certificates to the puppet-admin certificate allowlist.

To add allowed certificates using `pe.conf`:
1. Add the following code to your `pe.conf` file on your primary server and add the desired certificates.

```plaintext
puppet_enterprise::master::puppetserver::puppet_admin_certs:'example_cert_name'
```

2. Run `puppet agent -t`

**Disable update checking**

Puppet Server (pe-puppetserver) checks for updates when it starts or restarts, and every 24 hours thereafter. It transmits basic, anonymous info to our servers at Puppet by Perforce to get update information. You can optionally turn this off.

Specifically, it transmits:

- Product name
- Puppet Server version
- IP address
- Data collection timestamp

To turn off update checking using the console:

1. Open the console, click **Node groups**, and select the **PE Master** node group.
2. On the **Classes** tab, find the `puppet_enterprise::profile::master` class, add the `check_for_updates` parameter from the list, and change its value to `false`.
3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

**Puppet Server configuration files**

At startup, Puppet Server reads all of the `.conf` files in the `conf.d` directory (`/etc/puppetlabs/puppetserver/conf.d`).

The `conf.d` directory contains the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth.conf</td>
<td>Contains authentication rules and settings for agents and API endpoint access.</td>
</tr>
<tr>
<td>global.conf</td>
<td>Contains global configuration settings for Puppet Server, including logging settings.</td>
</tr>
<tr>
<td>metrics.conf</td>
<td>Contains settings for Puppet Server metrics services.</td>
</tr>
<tr>
<td>pe-puppet-server.conf</td>
<td>Contains Puppet Server settings specific to Puppet Enterprise.</td>
</tr>
<tr>
<td>webserver.conf</td>
<td>Contains SSL service configuration settings.</td>
</tr>
<tr>
<td>ca.conf</td>
<td>(Deprecated) Contains rules for Certificate Authority services. Superseded by <code>webserver.conf</code> and <code>auth.conf</code>.</td>
</tr>
</tbody>
</table>

For information about Puppet Server configuration files, see **Puppet Server's config files** for the Puppet Server version you're using, and the Related information links below.

**Related information**

**Viewing and managing Puppet Server metrics** on page 415

© 2022 Puppet, Inc., a Perforce company
Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

**pe-puppet-server.conf settings**

The `pe-puppet-server.conf` file contains Puppet Server settings specific to Puppet Enterprise, with all settings wrapped in a `jruby-puppet` section.

**enable-file-sync-locking or file_sync::file_sync_locking_enabled**

Controls whether the file sync client locks the JRuby pool (and, by extension, most requests to Puppet Server) while deploying Puppet code.

Default: `true`

⚠️ **CAUTION:** We do not recommend changing the `enable-file-sync-locking` setting. Instead, enable Lockless code deploys on page 781, which allow the file sync client to update code into versioned code directories without blocking Puppet Server requests or overwriting the live code directory.

**gem-home**

Determines where JRuby looks for gems. It is also used by the `puppetserver gem` command line tool.

Default: '/opt/puppetlabs/puppet/cache/jruby-gems'

**jruby_max_active_instances**

Controls the maximum number of JRuby instances to allow on the Puppet Server.

Default: 4

**master-conf-dir**

Sets the Puppet configuration directory's path.

Default: `/etc/puppetlabs/puppet`

**master-var-dir**

Sets the Puppet variable directory's path.

Default: `/opt/puppetlabs/server/data/puppetserver`

**max_requests_per_instance**

Sets the maximum number of requests allowed for each JRuby interpreter instance before it is killed.

Default: 100000

**max-queued-requests**

Sets the maximum number of requests allowed to be queued waiting to borrow from the JRuby pool.

This setting is optional and defaults to 0 (unlimited). If set, specify a positive integer.

After reaching the limit, all new requests receive a 503 Service Unavailable response until the queue drops below the limit. If the `max-retry-delay` setting is also set to a positive integer, then the 503 response includes a random sleep time after which the client can retry the request.

Don't use this setting if your managed infrastructure includes multiple agents older than Puppet 5.3. Because older agents treat 503 responses as failures, a thundering herd problem occurs when the agents schedule their next runs at the same time.

**max-retry-delay**

Sets the maximum number of seconds allowed for the random sleep time set when the `max-queued-requests` limit is exceeded. The random sleep time is returned as a Retry-After header on the 503 response for each rejected request.

Default: 1800 seconds

© 2022 Puppet, Inc., a Perforce company
If `max-queued-requests` is 0, there is no limit to the number of queued requests and, therefore, the `max-retry-delay` is irrelevant.

**pre-commit-hook-commands or puppetserver::pre_commit_hook_commands**

Specify scripts, as an array of strings, that you want the file sync storage server to execute against a repo after receiving a change but before committing and syncing the change across compilers. This is similar to Configuring `postrun commands` on page 823 for r10k or Configuring `post-environment hooks` on page 784 for Code Manager.

Scripts are executed in the order supplied by the `pe-puppetserver` process as the `pe-puppet` user. You must supply scripts as absolute paths. Additionally, the `pe-puppet` user must be able to execute the scripts, and the scripts must be able to consume `stdin` (even if the script doesn't do anything with it).

Default: `[/opt/puppetlabs/server/bin/generate-puppet-types.rb]`

⚠️ **CAUTION:** Removing the default value disables the `generate-puppet-types.rb` script. Unless you intentionally want to disable this type generation method, make sure to keep the default path in the array.

The output, error, and exit code for these scripts are logged at the trace level of `pe-puppetserver` logs when the exit code is 0. If the exit code is not 0, the codes are logged at the error level.

This setting is managed in PE modules, and you can override it by setting the `puppetenterprise::master::puppetserver::pre_commit_hook_commands` parameter in Hiera. Make sure to include the default path (for `generate-puppet-types.rb`) to ensure custom type are correctly cached. If you want to disable all pre-commit commands, supply an empty array in Hiera.

**puppet-code-repo**

Identifies, as a string, the internal name for the Puppet code repo (the codedir) that contains all code to sync across compilers (including user-supplied code repos).

⚠️ **CAUTION:** Do not change this setting.

Default: `'puppet-code'`

**ruby-load-path**

Sets the Puppet configuration directory's path. The agent's `libdir` path is included by default.

Default: `'/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib'`

---

## Configuring PuppetDB

After you've installed Puppet Enterprise, optimize it for your environment by configuring PuppetDB as needed.

This page covers a few key topics, but additional settings and information about configuring PuppetDB is available in the PuppetDB configuration documentation. Be sure to check that the PuppetDB docs version you're looking at matches the one version of PuppetDB in your PE.

### Configure agent run reports

By default, every time Puppet runs, the primary server generates agent run reports and submits them to PuppetDB. You can enable or disable this as needed.

To enable or disable agent run reports using the console:

1. Click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
2. On the **Classes** tab, add the `puppetenterprise::profile::master::puppetdb` class, select the `report_processor_ensure` parameter, and enter the value `present` to enable agent run reports or `absent` to disable agent run reports.
3. Click **Add parameter** and commit changes.

4. On the nodes hosting the primary server and console, run **Puppet**.

### Configure how long before PE stops managing deactivated nodes

Use the `node-purge-ttl` parameter to set the length of time before PE automatically removes deactivated or expired nodes. Once the time limit expires, nodes and their relevant facts, catalogs, and reports are removed from the PuppetDB only. Agent certificates on the Certificate Authority (CA) server are untouched.

To use the console to change the amount of time before nodes are purged:

1. Click **Node groups**, and select the **PE PuppetDB** group in the PE Infrastructure group.

2. On the **Classes** tab, find the `puppetenterprise::profile::puppetdb` class and the `node_purge_ttl` parameter, and change its value to the desired amount of time.

   Use these suffixes to change the unit of time:
   - Days: `d`
   - Hours: `h`
   - Minutes: `m`
   - Seconds: `s`
   - Milliseconds: `ms`

   For example, to set the purge time to 14 days:

   ```
   puppetenterprise::profile::puppetdb::node_purge_ttl: '14d'
   ```

3. Click **Add parameter** and commit changes.

4. Run **Puppet** on the nodes hosting the primary server and console.

### Change the PuppetDB user password

The console uses a database user account to access its PostgreSQL database. Change it if it is compromised or to comply with security guidelines.

To change the password:

1. Stop the `pe-puppetdb` puppet service by running `puppet resource service pe-puppetdb ensure=stopped`

2. On the database server (which might or might not be the same as PuppetDB, depending on your deployment's architecture), use the PostgreSQL administration tool of your choice to change the user's password. With the standard PostgreSQL client, you can do this by running `ALTER USER console PASSWORD '<new password>';`

3. Edit `/etc/puppetlabs/puppetdb/conf.d/database.ini` on the PuppetDB server and change the password: line under `common` or `production`, depending on your configuration, to contain the new password.

4. Start the `pe-puppetdb` service on the console server by running `puppet resource service pe-puppetdb ensure=running`

### Configure excluded facts

Use the `facts_blacklist` parameter to exclude facts from being stored in the PuppetDB database.

To use Hiera to specify facts you want to exclude:
1. Add the following to your default `.yaml` file and list the facts you want to exclude. For example, to exclude the facts `system_uptime_example` and `mountpoints_example`:

   ```yaml
   puppet_enterprise::puppetdb::database_ini::facts_blacklist:
   - 'system_uptime_example'
   - 'mountpoints_example'
   ```

2. Run `puppet agent -t` to compile changes.

### Configuring security settings

Ensure your PE environment is secure by configuring security settings.

#### Configure cipher suites

Due to regulatory compliance or other security requirements, you may need to change which cipher suites your SSL-enabled PE services use to communicate with other PE components. See Compatible ciphers on page 13 to ensure your ciphers are compatible with PE.

To add or remove cipher suites for certain types of services, use Hiera to add an array of ciphers to these parameters:

- **puppet_enterprise::ssl_cipher_suites**
  
  Set an array of ciphers, in IANA format, for all Java-based services: PuppetDB, Puppet Server, console services, and the orchestrator.

  ```yaml
  puppet_enterprise::ssl_cipher_suites:
  - 'TLS_AES_256_GCM_SHA384'
  - 'TLS_AES_128_GCM_SHA256'
  - 'TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256'
  - 'TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384'
  ```

- **puppet_enterprise::ssl_cipher_suites_non_java**

  Set an array of ciphers, in OpenSSL format, for all non-Java services: Bolt Server, ACE Server, PostgreSQL.

  ```yaml
  puppet_enterprise::ssl_cipher_suites_non_java:
  - 'ECDHE-ECDSA-AES128-GCM-SHA256'
  - 'ECDHE-ECDSA-AES256-GCM-SHA384'
  - 'ECDHE-RSA-AES128-GCM-SHA256'
  - 'ECDHE-RSA-AES256-GCM-SHA384'
  ```

- **puppet_enterprise::ssl_cipher_suites_browser**

  Set an array of ciphers, in OpenSSL format, accepted by the PE console in the browser (NGINX).

  ```yaml
  puppet_enterprise::ssl_cipher_suites_browser:
  - 'TLS_CHACHA20_POLY1305_SHA256'
  - 'ECDHE-ECDSA-CHACHA20-POLY1305'
  - 'ECDHE-RSA-CHACHA20-POLY1305'
  - 'ECDHE-ECDSA-AES128-GCM-SHA256'
  ```

#### Configure SSL protocols

Add or remove SSL protocols in your PE infrastructure.

To change what SSL protocols your PE infrastructure uses, use Hiera or the console to add or remove protocols.
Use the parameter `puppet_enterprise::master::puppetserver::ssl_protocols` and add an array for protocols you want to include, or remove protocols you no longer want to use.

For example, to enable TLSv1.1 and TLSv1.2, set the following parameter in the PE Infrastructure group in the console or in your Hiera data.

```bash
puppet_enterprise::master::puppetserver::ssl_protocols: ["TLSv1.3", "TLSv1.2"]
```

**Note:** To comply with security regulations, PE 2019.8.9 and later uses only version 1.2 and 1.3 of the Transport Layer Security (TLS) protocol.

### Configure RBAC and token-based authentication settings

Tune RBAC and token-based authentication settings, like setting the number of failed attempts a user has before they are locked out of the console or changing the amount of time a token is valid for.

RBAC and token authentication settings can be changed in the PE Infrastructure group in the console or in your Hiera data. Below is a list of related settings.

**puppet_enterprise::profile::console::rbac_failed_attempts_lockout**

An integer specifying how many failed login attempts are allowed on an account before the account is revoked. The default is "10" (attempts).

**puppet_enterprise::profile::console::rbac_password_reset_expiration**

An integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user to that they can reset their password. The default is "24" (hours).

**puppet_enterprise::profile::console::rbac_session_timeout**

Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console. The default is "60" (minutes).

**puppet_enterprise::profile::console::rbac_token_auth_lifetime**

A value representing the default authentication lifetime for a token. It cannot exceed the `rbac_token_maximum_lifetime`. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "1h".

**puppet_enterprise::profile::console::rbac_token_maximum_lifetime**

A value representing the maximum allowable lifetime for all tokens. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "10y".

**puppet_enterprise::profile::console::rbac_account_expiry_check_minutes**

An integer that specifies, in minutes, how often the application checks for idle user accounts. The default value is "60" (minutes).

**puppet_enterprise::profile::console::rbac_account_expiry_days**

An integer that specifies, in days, the duration before an inactive user account expires. The default is undefined. To activate the feature, add a value of "1" or greater.

If a non-superuser hasn't logged into the console during this specified period, their user status updates to revoked. After creating an account, if a non-superuser hasn't logged in to the console during the specified period, their user status updates to revoked.

**Note:** If you do not specify the `rbac_account_expiry_days` parameter, the `rbac_account_expiry_check_minutes` parameter is ignored.

### Related information

**RBAC API** on page 353

Use the RBAC API v1 and v2 endpoints to manage users, roles, tokens, password, and LDAP connections.

**Console and console-services parameters** on page 216

© 2022 Puppet, Inc., a Perforce company
Use these parameters to customize the behavior of the console and console-services. Parameters that begin with `puppet_enterprise::profile` can be modified from the console itself. See the configuration methods documents for more information on how to change parameters in the console or Hiera.

**RBAC database configuration**

Credential information for the RBAC service is stored in a PostgreSQL database.

The configuration information for that database is found in the 'rbac-database' section of the config.

For example:

```json
rbac-database: {
    classname: org.postgresql.Driver
    subprotocol: postgresql
    subname: "//<path-to-host>:5432/perbac"
    user: <username here>
    password: <password here>
}
```

**classname**

Used by the RBAC service for connecting to the database; this option should always be `org.postgresql.Driver`.

**subprotocol**

Used by the RBAC service for connecting to the database; this option should always be `postgresql`.

**subname**

JDBC connection path used by the RBAC service for connecting to the database. This should be set to the hostname and configured port of the PostgreSQL database. `perbac` is the database the RBAC service uses to store credentials.

**user**

This is the username the RBAC service should use to connect to the PostgreSQL database.

**password**

This is the password the RBAC service should use to connect to the PostgreSQL database.

---

**Add a custom HSTS header to NGINX**

An HTTP Strict Transport Security response header (HSTS) is a security header that blocks access to non-HTTPS content, preventing the browser from being exploited by man-in-the-middle attacks. HSTS headers do not allow the use of self-signed certificates, so if you want to enable an HSTS header, you must first add a custom certificate with a trusted CA to the console. Then, use a custom NGINX module to define and manage the header content.

**Before you begin**

Add a custom certificate. See [Use a custom SSL certificate for the console](#).
1. Write a custom NGINX module for managing the headers that includes a headers.conf file. For example, the custom module nginxcustom has this location and structure:

```
/etc/puppetlabs/code/environments/production/modules/nginxcustom
```

```
### files
#       ### headers.conf
### manifests
### init.pp
```

To learn about writing custom modules, see the Beginner's guide to writing modules.

2. Define the header content in the header.conf file. For example:

```
# cat files/headers.conf
#Custom headers
add_header X-Frame-Options SAMEORIGIN always;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection '1; mode=block';
add_header Strict-Transport-Security 'max-age=63072000; includeSubDomains; preload';
```

3. In manifests/init.pp, create a file resource that points to the module you're using to manage your headers.conf file. For example, to point to the nginxcustom module:

```
# cat manifests/init.pp
#File resource to create headers.conf
class nginxcustom {  
    file { '/etc/puppetlabs/nginx/headers.conf':  
        ensure => file,  
        group => 'root',  
        owner => 'root',  
        mode => '0644',  
        source =>'puppet:///modules/nginxcustom/headers.conf',
        }
}
```

4. In manifests/init.pp, use pe_nginx::directive to include the headers.conf file in the proxy.conf configuration file, allowing the headers to appear in the NGINX configuration in PE.

```
#Nginx directive to add include statement
pe_nginx::directive { 'include custom headers':  
    directive_ensure => 'present',  
    target => '/etc/puppetlabs/nginx/conf.d/proxy.conf',  
    directive_name => 'include',  
    value => 'headers.conf',  
    server_context => $::fqdn, }
```

5. Run Puppet: puppet agent -t

Related information

Use a custom SSL certificate for the console on page 838

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.
Configuring proxies

You can work around limited internet access by configuring proxies at various points in your infrastructure, depending on your connectivity limitations.

The examples provided here assume an unauthenticated proxy running at `proxy.example.vlan` on port 8080.

Downloading agent installation packages through a proxy

If your primary server doesn't have internet access, it can't download agent installation packages. If you want to use package management to install agents, set up a proxy and specify its connection details so `pe_repo` can access agent tarballs.

In the `pe_repo` class of the PE Master node group, specify values for `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` settings.

If you want to specify these settings in `pe.conf`, add the following to your `pe.conf` file with your desired parameters:

```plaintext
"pe_repo::http_proxy_host": "proxy.example.vlan",
"pe_repo::http_proxy_port": 8080
```

Tip: To test proxy connections to `pe_repo`, run:

```plaintext
proxy_uri='http://proxy.example.vlan:8080'
uri='https://pm.puppetlabs.com'
curl --proxy "$proxy_uri" --head "$uri"
```

See Usage notes for curl examples for information about forming curl commands.

Setting a proxy for agent traffic

General proxy settings in `puppet.conf` manage HTTP connections that are directly initiated by the agent.

To configure agents to communicate through a proxy using `pe.conf`, specify values for the `http_proxy_host` and `http_proxy_port` settings in `/etc/puppetlabs/puppet/puppet.conf`. For example:

```plaintext
http_proxy_host = proxy.example.vlan
http_proxy_port = 8080
```

For more information about HTTP proxy host options, including `no_proxy`, see the `http_proxy_host` entry in the Puppet Configuration Reference.

Setting a proxy for Code Manager traffic

Code Manager has its own set of proxy configuration options you can use to set a proxy for connections to the Git server or the Forge. These settings are unaffected by the proxy settings in `puppet.conf` because Code Manager is run by Puppet Server.

Note: To set a proxy for Code Manager connections, you must use an HTTP URL for your r10k remote and for all Puppetfile module entries.

Use a proxy for all HTTP connections, including Git and the Forge, when configuring Code Manager.
To use Hiera to configure Code Manager to use a proxy, add the following code to your default .yaml. Replace the example with your proxy name.

```yaml
puppet_enterprise::profile::master::r10k_proxy: "http://proxy.example.vlan:8080"
```

Tip: To test proxy connections to Git or the Forge, run one of these commands:

```bash
proxy_uri='http://proxy.example.vlan:8080'
uri='https://github.com'
curl --proxy "$proxy_uri" --head "$uri"

proxy_uri='http://proxy.example.vlan:8080'
uri='https://forgeapi.puppet.com'
curl --proxy "$proxy_uri" --head "$uri"
```

See Usage notes for curl examples for information about forming curl commands.

For detailed information about configuring proxies for Code Manager traffic, see Managing code with Code Manager on page 774.

Related information

Configuring proxies on page 787

To configure proxy servers, use the proxy setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

Configuring the console

After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.

Configure the PE console and console-services

Configure the behavior of the console and console-services, as needed.

Note: Use the Hiera or pe.conf method to configure non-profile classes, such as puppet_enterprise::api_port and puppet_enterprise::console_services::no_longer_reporting_cutoff.

To configure settings in the console:

1. Click Node groups, and select the node group that contains the class you want to work with.
2. On the Classes tab, find the class you want to work with, select the Parameter name from the list and edit its value.
3. Click Add parameter and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Related information

Configuring Puppet orchestrator on page 614

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

Running Puppet on nodes on page 458
Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

**Running Puppet on demand** on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

### Console and console-services parameters

Use these parameters to customize the behavior of the console and console-services. Parameters that begin with `puppet_enterprise::profile` can be modified from the console itself. See the configuration methods documents for more information on how to change parameters in the console or Hiera.

**puppet_enterprise::profile::console::classifier_synchronization_period**

Integer representing, in seconds, the classifier synchronization period, which controls how long it takes the node classifier to retrieve classes from the primary server.

Default: 600 (seconds).

**puppet_enterprise::profile::console::rbac_failed_attempts_lockout**

Integer specifying how many failed login attempts are allowed on an account before that account is revoked.

Default: 10 (attempts).

**puppet_enterprise::profile::console::rbac_password_reset_expiration**

Integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user so that they can reset their password.

Default: 24 (hours).

**puppet_enterprise::profile::console::rbac_session_timeout**

Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console.

Default: 60 (minutes).

**puppet_enterprise::profile::console::session_maximum_lifetime**

Integer representing the maximum allowable period that a console session can be valid. To not expire before the maximum token lifetime, set to '0'.

Supported units are "s" (seconds), "m" (minutes), "h" (hours), "d" (days), "y" (years). Units are specified as a single letter following an integer, for example "1d" (1 day). If no units are specified, the integer is treated as seconds.

**puppet_enterprise::profile::console::rbac_token_auth_lifetime**

A value representing the default authentication lifetime for a token. It cannot exceed the `rbac_token_maximum_lifetime`. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "1h".

**puppet_enterprise::profile::console::rbac_token_maximum_lifetime**

A value representing the maximum allowable lifetime for all tokens. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "10y".

**puppet_enterprise::profile::console::console_ssl_listen_port**

Integer representing the port that the console is available on.

Default: 443.

**puppet_enterprise::profile::console::ssl_listen_address**

Nginx listen address for the console.

Default: "0.0.0.0"
**puppet_enterprise::profile::console::classifier_prune_threshold**

An integer representing the number of days to wait before pruning the node classifier database. The node classifier database contains node check-in history if `classifier_node_check_in_storage` is enabled.

Set the value to 0 to never prune the node classifier database.

Default: 7 (days), but only has data to prune if `classifier_node_check_in_storage` is `true`.

**puppet_enterprise::profile::console::classifier_node_check_in_storage**

A Boolean specifying whether to create records when nodes check in with the node classifier. These records describe how nodes match the node groups they're classified into.

Set to `true` to enable node check-in storage.

Set to `false` to disable node check-in storage.

Default: `false`

**puppet_enterprise::profile::console::display_local_time**

A Boolean indicating whether to show timestamps in the local time or UTC.

Set to `true` to display timestamps in local time with hover text showing the equivalent UTC time.

Set to `false` to show timestamps in UTC time with no hover text.

Default: `false`

Modify these configuration parameters in Hiera or `pe.conf`, not the console:

**puppet_enterprise::api_port**

SSL port that the node classifier is served on.

Default: 4433

**puppet_enterprise::console_services::no_longer_reporting_cutoff**

Length of time, in seconds, before a node is considered unresponsive.

Default: 3600 (seconds)

**console_admin_password**

The password to log into the console, for example "myconsolepassword".

Default: Specified during installation.

### Related information

Configure RBAC and token-based authentication settings on page 295

Tune RBAC and token-based authentication settings, like setting the number of failed attempts a user has before they are locked out of the console or changing the amount of time a token is valid for.

### Manage the HTTPS redirect

By default, the console redirects to HTTPS when you attempt to connect over HTTP. You can customize the redirect target URL or disable redirection.

**Customize the HTTPS redirect target URL**

By default, the redirect target URL is the same as the FQDN of your primary server, but you can customize this redirect URL.

To change the target URL in the console:

1. Click **Node groups**, and select the **PE Infrastructure** node group.
2. On the **Classes** tab, find the `puppet_enterprise::profile::console::proxy::http_redirect` class, select the `server_name` parameter from the list, and change its value to the desired server.

3. Click **Add parameter** and commit changes.

4. On the nodes hosting the primary server and console, run Puppet.

**Disable the HTTPS redirect**
The pe-nginx webserver listens on port 80 by default. If you need to run your own service on port 80, you can disable the HTTPS redirect.

1. Add the following to your default `.yaml` file with the parameter set to `false`.

   ```yaml
   puppet_enterprise::profile::console::proxy::http_redirect::enable_http_redirect: false
   ```

2. To compile changes, run `puppet agent -t` on the primary server.

**Enable data editing in the console**
The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

1. On your primary server, edit `/etc/puppetlabs/puppet/hiera.yaml` to add:

   ```yaml
   hierarchy:
   - name: "Classifier Configuration Data"
     data_hash: classifier_data
   ```

   Place any additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same hierarchy key, preferably below the `Classifier Configuration Data` entry.

   **Note:** If you enable data editing in the console, add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

2. If your environment is configured for high availability or has compilers, update `hiera.yaml` on your replica and compilers, respectively.

**Add custom PQL queries to the console**
Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary server, copy the `custom_pql_queries.json.example` file and remove the `.example` suffix.

   ```bash
   sudo cp /etc/puppetlabs/console-services/custom_pql_queries.json.example /etc/puppetlabs/console-services/custom_pql_queries.json
   ```

2. Edit the file contents to include your own PQL queries or remove any existing queries.

3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop down options when running jobs.

**Configuring orchestration**
After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.

**Related information**

- [How Puppet orchestrator works](#) on page 604
With the Puppet orchestrator, you can run Puppet, tasks, or plans on-demand.

Configure the orchestrator and `pe-orchestration-services`

These are some optional parameters you can use to configure the behavior of the orchestrator and the `pe-orchestration-services` service.

You can modify these profile class parameters in the Puppet Enterprise (PE) console on the Classes tab for the PE Orchestrator infrastructure node group.

**`puppet_enterprise::profile::orchestrator::task_concurrency`**

Integer representing the number of simultaneous task or plan actions that can run at the same time in the orchestrator. All task and plan actions are limited by this concurrency limit regardless of transport type (WinRM, SSH, PCP).

If a task or plan action runs on multiple nodes, each node consumes one action. For example, if a task needs to run on 300 nodes, and your `task_concurrency` is set to 200, then the task can run on 200 nodes while the remaining 100 nodes wait in queue.

Default: 250 (actions)

**`puppet_enterprise::profile::bolt_server::concurrency`**

An integer that determines the maximum number of simultaneous task or plan requests orchestrator can make to bolt-server. Only task or plan executions on nodes with SSH or WinRM transport methods are limited by this setting because only they require requests to bolt-sever.

Default: 100 (requests)

**CAUTION:** Do not set a concurrency limit that is higher than the bolt-server limit. This can cause timeouts that lead to failed task runs.

**`puppet_enterprise::profile::agent::pxp_enabled`**

Disable or enable the PXP service by setting it to `true` or `false`. If you disable this setting you can’t use the orchestrator or the Run Puppet button in the console.

Default: `true`

**`puppet_enterprise::profile::orchestrator::global_concurrent_compiles`**

An integer that determines how many concurrent compile requests can be outstanding to the primary server, across all orchestrator jobs.

Default: 8 (requests)

**`puppet_enterprise::profile::orchestrator::job_prune_threshold`**

An integer of 2 or greater, which specifies the number of days to retain job reports.

This parameter sets the corresponding parameter `job-prune-days-threshold`.

While `job_prune_threshold` itself has no default value, `job-prune-days-threshold` has a default of 30 (30 days).

**`puppet_enterprise::profile::orchestrator::pcp_timeout`**

An integer that represents how many seconds must pass while an agent attempts to connect to a PCP broker. If the agent can’t connect to the broker in that time frame, the run times out.

Default: 30 (seconds)

**`puppet_enterprise::profile::orchestrator::run_service`**

Disable or enable orchestration services. Set to `true` or `false`.

Default: `true`

**`puppet_enterprise::profile::orchestrator::allowed_pcp_status_requests`**

An integer that defines how many times an orchestrator job allows status requests to time out before a job is considered failed. Status requests wait 12 seconds between timeouts, so multiply the value of the
allowed_pcp_status_requests by 12 to determine how many seconds the orchestrator waits on targets that aren’t responding to status requests.

Default: 35 (timeouts)

**puppet_enterprise::profile::orchestrator::java_args**

Specifies the heap size, or, the amount of memory that each Java process is allowed to request from the operating for the orchestrator to use.

Default: 704 MB

**puppet_enterprise::profile::orchestrator::jruby_max_active_instances**

An integer that determines the maximum number of JRuby instances that the orchestrator creates to execute plans. Because each plan uses one JRuby to run, this value is effectively the maximum number of concurrent plans. Setting the orchestrator heap size (java_args) automatically sets the jruby_max_active_instances using the formula java_args / 1024. If the value equals less than one, the default is one JRuby instance.

Default: 1 (instance)

**Note:** The jruby_max_active_instances pool for the orchestrator is separate from the Puppet Server pool. See the JRuby max active instances on page 282 tuning guide for more information.

**puppet_enterprise::profile::plan_executor::versioned_deploys**

Set to true to enable versioned deployments of environment code. Use this setting for Running plans alongside code deployments on page 669.

Default: false

**Important:** Setting this to true disables the file sync client's locking mechanism that usually enforces a consistent environment state for your plans. This can cause Puppet functions and plans that call other plans to behave unexpectedly if a code deployment occurs while a plan is running.

For information about how the orchestrator works, what you can do with it, and additional parameters and configuration options, refer to Orchestrating Puppet runs, tasks, and plans on page 603.

**Related information**

**Configuring Puppet orchestrator** on page 614

Once you’ve installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

**Running Puppet on nodes** on page 458

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

**Running Puppet on demand** on page 620

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

**Configure the PXP agent**

Puppet Execution Protocol (PXP) is a messaging system used to request tasks and communicate task statuses. The PXP agent runs the PXP service and you can configure it using Hiera or the console.

**puppet_enterprise::pxp_agent::ping_interval**

Controls how frequently (in seconds) PXP agents will ping PCP brokers. If the brokers don’t respond, the agents try to reconnect.

Default: 120 (seconds)

**puppet_enterprise::pxp_agent::pxp_logfile**

A string that represents the path to the PXP agent log file and can be used to debug issues with orchestrator.
Default:
- *nix: `/var/log/puppetlabs/pxp-agent/pxp-agent.log`
- Windows: `C:\Program Data\PuppetLabs\pxp-agent\var\log\pxp-agent.log`

**puppet_enterprise::pxp_agent::spool_dir_purge_ttl**
The amount of time to keep records of old Puppet or task runs on agents. You can declare time in minutes (30m), hours (2h), and days (14d).
Default: 14d

**puppet_enterprise::pxp_agent::task_cache_dir_purge_ttl**
Controls how long tasks are cached after use. You can declare time in minutes (30m), hours (2h), and days (14d).
Default: 14d

**puppet_enterprise::pxp_agent::broker_proxy**
Sets a proxy URI used to connect to the pcp-broker to listen for task and Puppet runs.

**puppet_enterprise::pxp_agent::master_proxy**
Sets a proxy URI used to connect to the primary server to download task implementations.

**puppet_enterprise::pcp_max_message_size_mb**
Sets the message size, in mb, for pcp_broker, pxp_agent, and the orchestrator. The maximum message size cannot be higher than the default of 64mb, so you can only reduce it.
Default: 64 (mb)

**Note:** We do not recommend changing the `pcp_max_message_size_mb` parameter if you send or receive large payloads because it might cause errors for large task and plan run parameters and output.

### Correct ARP table overflow

In larger deployments that use the PCP broker, you might encounter ARP table overflows and need to adjust some system settings.

Overflows occur when the ARP table—a local cache of IP address to MAC address resolutions—fills and starts evicting old entries. When frequently used entries are evicted, network traffic will increase to restore them, increasing network latency and CPU load on the broker.

A typical log message looks like:

```
[root@s1 peadmin]# tail -f /var/log/messages
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
```

To work around this issue:

Increase sysctl settings related to ARP tables.

For example, the following settings are appropriate for networks hosting up to 2000 agents:

```
# Set max table size
net.ipv6.neigh.default.gc_thresh3=4096
net.ipv4.neigh.default.gc_thresh3=4096
# Start aggressively clearing the table at this threshold
net.ipv6.neigh.default.gc_thresh2=2048
net.ipv4.neigh.default.gc_thresh2=2048
# Don't clear any entries until this threshold
net.ipv6.neigh.default.gc_thresh1=1024
net.ipv4.neigh.default.gc_thresh1=1024
```
Configuring ulimit

As your infrastructure grows and you bring more agents under management, you might need to increase the number allowed file handles per client.

Configure ulimit for PE services

The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults are not adequate for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

You can increase the limits for the following services:

• pe-orchestration-services
• pe-puppetdb
• pe-console-services
• pe-puppetserver
• pe-puppet

The location and method for configuring ulimit depends on your agent’s platform. You might use systemd, upstart, or some other init system.

In the following instructions, replace `<PE SERVICE>` with the specific service you’re editing. The examples show setting a limit of 32768, which you can also change according to what you need.

Configure ulimit using systemd

With systemd, the number of open file handles allowed is controlled by a setting in the service file at `/usr/lib/systemd/system/<PE SERVICE>.service`.

1. To increase the limit, run the following commands, setting the `LimitNOFILE` value to the new number:

   ```
   mkdir /etc/systemd/system/<PE SERVICE>.service.d
   echo "[Service] LimitNOFILE=32768" > /etc/systemd/system/<PE SERVICE>.service.d/limits.conf
   systemctl daemon-reload
   ```

2. Confirm the change by running: `systemctl show <PE SERVICE> | grep LimitNOFILE`

Configure ulimit using upstart

For Ubuntu and Red Hat systems, the number of open file handles allowed for is controlled by settings in service files.

The service files are:

• Ubuntu: `/etc/default/<PE SERVICE>`
• Red Hat: `/etc/sysconfig/<PE SERVICE>`

For both Ubuntu and Red Hat, set the last line of the file as follows:

`ulimit -n 32678`

This sets the number of open files allowed at 32,678.

Configure ulimit on other init systems

The ulimit controls the number of processes and file handles that the PE service user can open and process.

To increase the ulimit for a PE service user:

Edit `/etc/security/limits.conf` so that it contains the following lines:

```
<PE SERVICE USER> soft nofile 32768
```
Writing configuration files

Puppet supports two formats for configuration files that configure settings: valid JSON and Human-Optimized Config Object Notation (HOCON), a JSON superset.

For more information about HOCON itself, see the HOCON documentation.

Configuration file syntax

Refer to these examples when you're writing configuration files to identify correct JSON or HOCON syntax.

Brackets

In HOCON, you can omit the brackets ( { } ) around a root object.

<table>
<thead>
<tr>
<th>JSON example</th>
<th>HOCON example</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>&quot;authorization&quot;: {</td>
</tr>
<tr>
<td>&quot;authorization&quot;: {</td>
<td>&quot;version&quot;: 1</td>
</tr>
<tr>
<td>&quot;version&quot;: 1</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Quotes

In HOCON, double quotes around key and value strings are optional in most cases. However, double quotes are required if the string contains the characters *, ^, +, :, or =.

<table>
<thead>
<tr>
<th>JSON example</th>
<th>HOCON example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;authorization&quot;: {</td>
<td>authorization: {</td>
</tr>
<tr>
<td>&quot;version&quot;: 1</td>
<td>version: 1</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>
Commas

When writing a map or array in HOCON, you can use a new line instead of a comma.

<table>
<thead>
<tr>
<th>JSON example</th>
<th>HOCON example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>Map</td>
</tr>
<tr>
<td>rbac: {</td>
<td>rbac: {</td>
</tr>
<tr>
<td>password-reset-expiration: 24,</td>
<td>password-reset-expiration: 24</td>
</tr>
<tr>
<td>session-timeout: 60,</td>
<td>session-timeout: 60</td>
</tr>
<tr>
<td>failed-attempts-lockout: 10,</td>
<td>failed-attempts-lockout: 10</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Array

<table>
<thead>
<tr>
<th>JSON example</th>
<th>HOCON example</th>
</tr>
</thead>
<tbody>
<tr>
<td>http-client: {</td>
<td>http-client: {</td>
</tr>
</tbody>
</table>
|   ssl-protocols: [TLSv1, TLSv1.1, TLSv1.2] |   ssl-protocols: [
| }             | TLSv1 |
|               | TLSv1.1 |
|               | TLSv1.2  |
| }             | }         |

Comments

Add comments using either // or #. Inline comments are supported.

Analytics data collection

Some components automatically collect data about how you use Puppet Enterprise. If you want to opt out of providing this data, you can do so, either during or after installing.

What data does Puppet Enterprise collect?

Puppet Enterprise (PE) collects the following data when Puppet Server starts or restarts, and again every 24 hours.

License, version, primary server, and agent information:

- License UUID
- Number of licensed nodes
- Product name
- PE version
• Primary server's operating system
• Primary server's public IP address
• Whether the primary server is running on Microsoft Azure
• The hypervisor the primary server is running on, if applicable
• Number of nodes in deployment
• Agent operating systems
• Number of agents running each operating system
• Agent versions
• Number of agents running each version of Puppet agent
• All-in-One (AIO) puppet-agent package versions
• Number of agents running on Microsoft Azure or Google Cloud Platform, if applicable
• Number of configured disaster recovery replicas, if applicable

Puppet Enterprise feature use information:
• Number of node groups in use
• Number of nodes used in orchestrator jobs after last orchestrator restart
• Mean nodes per orchestrator job
• Maximum nodes per orchestrator job
• Minimum nodes per orchestrator job
• Total orchestrator jobs created after last orchestrator restart
• Number of non-default user roles in use
• Type of certificate autosigning in use
• Number of nodes in the job that were run over Puppet Communications Protocol
• Number of nodes in the job that were run over SSH
• Number of nodes in the job that were run over WinRM
• Number of nodes patched per task run
• Type of operating system on nodes patched in a task run
• Number of patches applied to each node per task run
• Number of patches completed per task run
• Number of nodes with the pe_patch module
• Number of nodes with the pe_patch module that require patching
• List of Puppet task jobs
• List of Puppet deploy jobs
• List of Puppet task jobs run by plans
• List of file upload jobs run by plans
• List of script jobs run by plans
• List of command jobs run by plans
• List of wait jobs run by plans
• How nodes were selected for the job
• Whether the job was started by the PE admin account
• Number of nodes in the job
• Length of description applied to the job
• Length of time the job ran
• User agent used to start the job (to distinguish between the console, command line, and API)
• UUID used to correlate multiple jobs run by the same user
• Time the task job was run
• How nodes were selected for the job
• Whether the job was started by the PE admin account
• Number of nodes in the job
• Length of description applied to the job
- Whether the job was asked to override agent-configured no-operation (no-op) mode
- Whether app-management was enabled in the orchestrator for this job
- Time the deploy job was run
- Type of version control system webhook
- Whether the request was to deploy all environments
- Whether code-manager will wait for all deploys to finish or error to return a response
- Whether the deploy is a dry-run
- List of environments requested to deploy
- List of deploy requests
- Total time elapsed for all deploys to finish or error
- List of total wait times for deploys specifying --wait option
- Name of environment deployed
- Time needed for r10k to run
- Time spent committing to file sync
- Time elapsed for all environment hooks to run
- List of individual environment deploys
- Puppet classes applied from publicly available modules, with node counts per class

Backup and Restore information:
- Whether user used --force option when running restore
- Scope of restore
- Time in seconds for various restore functions
- Time to check for disk space to restore
- Time to stop PE related services
- Time to restore PE file system components
- Time to migrate PE configuration for new server
- Time to configure PE on newly restored primary server
- Time to update PE classification for new server
- Time to deactivate the old primary server node
- Time to restore the pe-orchestrator database
- Time to restore the pe-rbac database
- Time to restore the pe-classifier database
- Time to restore the pe-activity database
- Time to restore the pe-puppetdb database
- Total time to restore
- List of puppet backup restore jobs
- Whether user used --force option when running puppet-backup create
- Whether user used --dir option when running puppet-backup create
- Whether user used --name option when running puppet-backup create
- Scope of backup
- Time in seconds for various back up functions
- Time needed to estimate backup size, disk space needed, and disk space available
- Time to create file system backup
- Time to back up the pe-orchestrator database
- Time to back up the pe-rbac database
- Time to back up the pe-classifier database
- Time to back up the pe-activity database
- Time to back up the pe-puppetdb database
- Time to compress archive file to backup directory
- Time to back up PE related classification

© 2022 Puppet, Inc., a Perforce company
• Total time to back up
• List of `puppet-backup create` jobs

Puppet Server performance information:

• Total number of JRuby instances
• Maximum number of active JRuby instances
• Maximum number of requests per JRuby instance
• Average number of instances not in use over the process’s lifetime
• Average wait time to lock the JRuby pool
• Average time the JRuby pool held a lock
• Average time an instance spent handling requests
• Average time spent waiting to reserve an instance from the JRuby pool
• Number of requests that timed out while waiting for a JRuby instance
• Amount of memory the JVM starts with
• Maximum amount of memory the JVM is allowed to request from the operating system

Installer information:

• Installation method (express, text, web, repair)
• Current version, if upgrading
• Target version
• Success or failure, and limited information on the type of failure, if an

If PE is installed using an Amazon Web Services Marketplace Image:

• The marketplace name
• Marketplace image billing mode (bring your own license or pay as you go)

While in use, the console collects the following information:

• Pageviews
• Link and button clicks
• Page load time
• User language
• Screen resolution
• Viewport size
• Anonymized IP address

The console does not collect user inputs such as node or group names, user names, rules, parameters, or variables.

The collected data is tied to a unique, anonymized identifier for each primary server and your site as a whole. No personally identifiable information is collected, and the data we collect is never used or shared outside Puppet by Perforce.

**How does sharing this data benefit you?**

We use the data to identify organizations that could be affected by a security issue, alert them to the issue, and provide them with steps to take or fixes to download. In addition, the data helps us understand how people use the product, which helps us improve the product to meet your needs.

**How does Puppet use the collected data?**

The data we collect is one of many methods we use for learning about our customers. For example, knowing how many nodes you manage helps us develop more realistic product testing. And learning which operating systems are the most and the least used helps us decide where to prioritize new functionality. By collecting data, we begin to understand you as a customer.
Opt out during the installation process
To opt out of data collection during installation, you can set the DISABLE_ANALYTICS environment variable when you run the installer script.

Setting the DISABLE_ANALYTICS environment variable during installation sets puppet_enterprise::send_analytics_data: false in pe.conf, opting you out of data collection.

Follow the instructions for your chosen installation method, adding DISABLE_ANALYTICS=1 when you call the installer script, for example:

```bash
sudo DISABLE_ANALYTICS=1 ./puppet-enterprise-installer
```

Opt out after installing
If you've already installed PE and want to disable data collection, follow these steps.

1. In the console, click Node groups, and then click PE Infrastructure.
2. On the Classes tab, in the puppet_enterprise class, add send_analytics_data as a parameter and set the Value to false.
3. On the Inventory page, select your primary server and click Run Puppet.
4. Select your console node and click Run Puppet.

After Puppet runs to enforce the changes on the master and console nodes, you have opted out of data collection.

Static catalogs
A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing source attributes pointing to puppet:// locations on a node.

The metadata in a static catalog can refer to a specific version of the file (not just the latest version), and can confirm that the agent is applying the appropriate version of the file resource for the catalog. Because the metadata is provided in the catalog, agents make fewer requests to the primary server.

See the open source Puppet documentation more information about Resources, File types, and Catalog compilation.

Enabling static catalogs
When a primary server produces a non-static catalog, the catalog doesn't specify the version of file resources. When the agent applies the catalog, it always retrieves the latest version of that file resource, or uses a previously retrieved version if it matches the latest version's contents.

This potential problem affects file resources that use the source attribute. File resources that use the content attribute are not affected, and their behavior does not change in static catalogs.

When a manifest depends on a file whose contents change more frequently than the agent receives new catalogs, a node might apply a version of the referenced file that doesn't match the instructions in the catalog.

Consequently, the agent's Puppet runs might produce different results each time the agent applies the same catalog. This often causes problems because Puppet generally expects a catalog to produce the same results each time it's applied, regardless of any code or file content updates on the primary server.

Additionally, each time an agent applies a normal cached catalog that contains file resources sourced from puppet:/// locations, the agent requests file metadata from the primary server each time the catalog's applied, even though nothing has changed in the cached catalog. This causes the primary server to perform unnecessary resource-intensive checksum calculations for each file resource.
Static catalogs avoid these problems by including metadata that refers to a specific version of the resource's file. This prevents a newer version from being incorrectly applied, and avoids having the agent regenerate the metadata on each Puppet run. The metadata is delivered in the form of a unique hash maintained, by default, by the file sync service.

We call this type of catalog "static" because it contains all of the information that an agent needs to determine whether the node's configuration matches the instructions and state of file resources at the static point in time when the catalog was generated.

**Differences in catalog behavior**

Without static catalogs enabled:

- The agent sends facts to the primary server and requests a catalog.
- The primary server compiles and returns the agent's catalog.
- The agent applies the catalog by checking each resource the catalog describes. If it finds any resources that are not in the desired state, it makes the necessary changes.

With static catalogs enabled:

- The agent sends facts to the primary server and requests a catalog.
- The primary server compiles and returns the agent's catalog, including metadata that that specifies the desired state of the node's file resources.
- The agent applies the catalog by checking each resource the catalog describes. If the agent finds any resources that are not in the desired state, it makes the necessary changes based on the state of the file resources at the static point in time when the catalog was generated.
- If you change code on the primary server, file contents are not updated until the agent requests a new catalog with new file metadata.

**Enabling file sync**

In PE, static catalogs are disabled across all environments for new installations. To use static catalogs in PE, you must enable file sync. After file sync is enabled, Puppet Server automatically creates static catalogs containing file metadata for eligible resources, and agents running Puppet 1.4.0 or newer can take advantage of the catalogs' new features.

If you do not enable file sync and Code Manager, you can still use static catalogs, but you need to create some custom scripts and set a few parameters in the console.

**Enforcing change with static catalogs**

When you are ready to deploy new Puppet code and deliver new static catalogs, you don’t need to wait for agents to check in. Use the Puppet orchestrator to enforce change and deliver new catalogs across your PE infrastructure, on a per-environment basis.

**When aren't static catalogs applied?**

In the following scenarios, either agents won't apply static catalogs, or catalogs won't include metadata for file resources.

- Static catalogs are globally disabled.
- Code Manager and file sync are disabled, and code_id and code_content aren't configured.
- Your agents aren't running PE 2016.1 or later (Puppet agent version 1.4.0 or later).

Additionally, Puppet won't include metadata for a file resource if it:

- Uses the `content` attribute instead of the `source` attribute.
- Uses the `source` attribute with a non-Puppet scheme (for example `source => 'http://host:port/path/to/file'`).
- Uses the `source` attribute without the built-in modules mount point.
• Uses the source attribute, but the file on the primary server is not in /etc/puppetlabs/code/environments/<environment>/*/*/files/**. For example, module files are typically in /etc/puppetlabs/code/environments/<environment>/modules/<module_name>/files/**.

Agents continue to process the catalogs in these scenarios, but without the benefits of inlined file metadata or file resource versions.

Related information

Enabling or disabling file sync on page 830
File sync is normally enabled or disabled automatically along with Code Manager.

How Puppet orchestrator works on page 604
With the Puppet orchestrator, you can run Puppet, tasks, or plans on-demand.

Disabling static catalogs globally with Hiera

You can turn off all use of static catalogs with a Hiera setting.

To disable static catalogs using Hiera:

1. Add the following code to your default .yaml file and set the parameter to false:

    puppet_enterprise::master::static_catalogs: false

2. To compile changes, run puppet agent -t

Using static catalogs without file sync

To use static catalogs without enabling file sync, you must set the code_id and code_content parameters in Puppet, and then configure the code_id_command, code_content_command, and file_sync_enabled parameters in the console.

1. Set code_id and code_content by following the instructions in the open source Puppet static catalogs documentation. Then return to this page to set the remaining parameters.
2. In the console, click Node groups, and in the PE Infrastructure node group, select the PE Master node group.
3. On the Classes tab, locate the puppet_enterprise::profile::master class, and select file_sync_enabled from its Parameter list.
4. In the Value field, enter false, and click Add parameter.
5. Select the code_id_command parameter, and in the Value field, enter the absolute path to the code_id script, and click Add parameter.
6. Select the code_content_command parameter, and in the Value field, add the absolute to the code_content script, and click Add parameter.
7. Commit changes.
8. Run Puppet on the primary server.

Related information

Configuring Puppet orchestrator on page 614
Once you’ve installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

Running Puppet on nodes on page 458
Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Running Puppet on demand on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

**Configuring disaster recovery**

Enabling disaster recovery for Puppet Enterprise ensures that your systems can fail over to a replica of your primary server if infrastructure components become unreachable.

- **Disaster recovery** on page 315
  Disaster recovery creates a replica of your primary server.

- **Configure disaster recovery** on page 324
  To configure disaster recovery, you must provision a replica to serve as backup during failovers. If your primary server is permanently disabled, you can then promote a replica.

**Disaster recovery**

Disaster recovery creates a replica of your primary server.

You can have only one replica at a time, and you can add disaster recovery to an installation with or without compilers. Disaster recovery isn't supported with FIPS-compliant installations.

There are two main advantages to enabling disaster recovery:

- If your primary server fails, the replica takes over, continuing to perform critical operations.
- If your primary server can't be repaired, you can promote the replica to primary server. Promotion establishes the replica as the new, permanent primary server.

⚠️ **CAUTION:** Provisioning a replica causes your primary server to consume disk space for unreplicated data. To ensure the continued health of your primary server, you must maintain syncing to your replica, or else promptly forget the replica.

**Disaster recovery architecture**

The replica is not an exact copy of the primary server. Rather, the replica duplicates specific infrastructure components and services. Hiera data and other custom configurations are not replicated.

Replication can be *read-write*, meaning that data can be written to the service or component on either the primary server or the replica, and the data is synced to both nodes. Alternatively, replication can be *read-only*, where data is written only to the primary server and synced to the replica. Some components and services, like Puppet Server and the console service UI, are not replicated because they contain no native data.

Some components and services are activated immediately when you enable a replica; others aren't active until you promote a replica. After you provision and enable a replica, it serves as a compiler, redirecting PuppetDB and cert requests to the primary server.

<table>
<thead>
<tr>
<th>Component or service</th>
<th>Type of replication</th>
<th>Activated when replica is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet Server</td>
<td>none</td>
<td>enabled</td>
</tr>
<tr>
<td>File sync client</td>
<td>read-only</td>
<td>enabled</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>read-write</td>
<td>enabled</td>
</tr>
<tr>
<td>Certificate authority</td>
<td>read-only</td>
<td>promoted</td>
</tr>
<tr>
<td>RBAC service</td>
<td>read-only</td>
<td>enabled</td>
</tr>
<tr>
<td>Node classifier service</td>
<td>read-only</td>
<td>enabled</td>
</tr>
<tr>
<td>Component or service</td>
<td>Type of replication</td>
<td>Activated when replica is...</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Activity service</td>
<td>read-only</td>
<td>enabled</td>
</tr>
<tr>
<td>Orchestration service</td>
<td>read-only</td>
<td>promoted</td>
</tr>
<tr>
<td>Console service UI</td>
<td>none</td>
<td>promoted</td>
</tr>
<tr>
<td>Agentless Catalog Executor (ACE)</td>
<td>none</td>
<td>promoted</td>
</tr>
<tr>
<td>service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bolt service</td>
<td>none</td>
<td>promoted</td>
</tr>
</tbody>
</table>

In a standard installation, when a Puppet run fails over, agents communicate with the replica instead of the primary server. In a large or extra-large installation with compilers, agents communicate with load balancers or compilers, which communicate with the primary server or replica.

**Related information**

Configure settings with Hiera on page 287

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

**What happens during failovers**

Failover occurs when the replica takes over services usually performed by the primary server.

Failover is automatic — you don’t have to take action to activate the replica. With disaster recovery enabled, Puppet runs are directed first to the primary server. If the primary server is either fully or partially unreachable, runs are directed to the replica.

In partial failovers, Puppet runs can use the server, node classifier, or PuppetDB on the replica if those services aren’t reachable on the primary server. For example, if the primary server’s node classifier fails, but its Puppet Server is still running, agent runs use the Puppet Server on the primary server but fail over to the replica’s node classifier.

What works during failovers:

- Scheduled Puppet runs
- Catalog compilation
- Viewing classification data using the node classifier API
- Reporting and queries based on PuppetDB data

What doesn’t work during failovers:

- Deploying new Puppet code
- Editing node classifier data
- Using the console
- Certificate functionality, including provisioning new agents, revoking certificates, or running the `puppet certificate` command
- Most CLI tools

**System and software requirements for disaster recovery**

Your Puppet infrastructure must meet specific requirements in order to configure disaster recovery.

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>All supported PE primary server platforms.</td>
</tr>
<tr>
<td>Component</td>
<td>Requirement</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Software        | • You must use Code Manager so that code is deployed to both the primary server and the replica after you enable a replica.  
• You must use the default PE node classifier so that disaster recovery classification can be applied to nodes.  
• Orchestrator must be enabled so that agents are updated when you provision or enable a replica. |
| Replica         | • Must be an agent node that doesn’t have a specific function already. You can decommission a node, uninstall all puppet packages, and re-commission the node to be a replica. However, a compiler cannot perform two functions, for example, as a compiler and a replica.  
• Must have the same hardware specifications and capabilities as your primary server. |
| Firewall        | • Your replica must comply with the same port requirements as your primary server to ensure that the replica can operate as the primary server during failover.  
• For details, see the firewall configuration requirements for your installation type. |
| Node names      | • You must use resolvable domain names when specifying node names for the primary server and replica. |
| RBAC tokens     | • You must have an admin RBAC token when running puppet infrastructure commands, including provision, enable, and forget. You can generate a token using the puppet-access command.  
Note: You don’t need an RBAC token to promote a replica. |

**Related information**

Firewall configuration on page 192
Follow these guidelines for firewall configuration based on your installation type.

Generate a token using puppet-access on page 349
Use puppet-access to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

**Classification changes in disaster recovery installations**

When you provision and enable a replica, the system makes a number of classification changes in order to manage disaster recovery.
Two infrastructure node groups are added in installations with disaster recovery. The **PE HA Master** node group includes your primary server and inherits from the **PE Master** node group. The **PE HA Replica** node group includes your replica and inherits from the **PE Infrastructure** node group.

Additional disaster recovery configuration is managed with these parameters:

- classifier_client_certname
- classifier_host
- classifier_port
- ha_enabled_replicas
- manage_puppet_conf
- pcp_broker_list
- primary_uris
- provisioned_replicas
- puppetdb_host
- puppetdb_port
- replica_hostnames
- replicating
- replication_mode
- server_list
- sync_allowlist
- sync_peers

**Note:** Apart from the parameters in the **PE Agent** and **PE Infrastructure Agent** node groups (manage_puppet_conf, server_list, pcp_broker_list, and primary_uris), all of these are system parameters that should not be manually modified. The **PE Agent** and **PE Infrastructure Agent** parameters are automatically updated based on the values you specify when you provision and enable a replica.

---

**classifier_client_certname**

**Purpose**

Specifies the name on the certificate used by the classifier.

**Node group**

PE Master

**Class**

puppet_enterprise::profile::master

**DR-only parameter**

No

**Example with enabled replica**

["<PRIMARY_CERTNAME>", "<REPLICA_CERTNAME>"

**Notes**

Replica values are appended to the end of parameter when a replica is enabled.

---

**classifier_host**

**Purpose**

Specifies the certname of the node running the classifier service.

**Node group**

PE Master
Class
  puppet_enterprise::profile::master

DR-only parameter
  No

Example with enabled replica
  ["<PRIMARY_CERTNAME>", "<REPLICA_CERTNAME>"]

Notes
  Replica values are appended to the end of parameter when a replica is enabled.

classifier_port

Purpose
  Specifies the port used for communicating with the classifier service. Always 4433.

Node group
  PE Master

Class
  puppet_enterprise::profile::master

DR-only parameter
  No

Example with enabled replica
  [4433, 4433]

Notes
  Replica values are appended to the end of parameter when a replica is enabled.

ha_enabled_replicas

Purpose
  Tracks replica nodes that are failover ready.

Node group
  PE Infrastructure

Class
  puppet_enterprise

DR-only parameter
  Yes

Example with enabled replica
  ["<REPLICA_CERTNAME>"]

Notes
  Updated when you enable a replica.

manage_puppet_conf

Purpose
  When true, specifies that the server_list setting is managed in puppet.conf.

Node group
  PE Agent, PE Infrastructure Agent
Class
   puppet_enterprise::profile::agent

DR-only parameter
   No

Example with enabled replica
   true

pcp_broker_list

Purpose
   Specifies the list of Puppet Communications Protocol brokers that Puppet Execution Protocol agents contact, in order.

Node group
   PE Agent, PE Infrastructure Agent

Class
   puppet_enterprise::profile::agent

DR-only parameter
   No

Example with enabled replica
   PE Agent — ["<PRIMARY_CERTNAME>:8142", "<REPLICA_CERTNAME>:8142"] or in a large installation, ["<LOAD_BALANCER>:8142"]
   PE Infrastructure Agent — ["<PRIMARY_CERTNAME>:8142", ","<REPLICA_CERTNAME>:8142"]

Notes
   • Infrastructure nodes must be configured to communicate directly with the primary in the PE Infrastructure Agent node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the PE Agent node group.
   • When a replica is enabled, the replica is appended to the end of the list in the PE Infrastructure Agent group, and when not using a load balancer, it's appended to the list in PE Agent.
   • Some puppet infrastructure commands refer to this parameter as agent-serverurls, but those commands nonetheless manage the server_list parameter.

   **Important:** Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported.

primary_uris

Purpose
   Specifies the list of Puppet Server nodes hosting task files for download that Puppet Execution Protocol agents contact, in order.

Node group
   PE Agent, PE Infrastructure Agent

Class
   puppet_enterprise::profile::agent

DR-only parameter
   No
Example with enabled replica

PE Agent — "<PRIMARY_CERTNAME>:8140," <REPLICA_CERTNAME>:8140"], or in a large installation, ["<LOAD_BALANCER>:8140"]

PE Infrastructure Agent — ["<PRIMARY_CERTNAME>:8140","<REPLICA_CERTNAME>:8140"]

Notes

• Infrastructure nodes must be configured to communicate directly with the primary in the PE Infrastructure Agent node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the PE Agent node group.

• When a replica is enabled, the replica is appended to the end of the list in the PE Infrastructure Agent group, and when not using a load balancer, it's appended to the list in PE Agent.

• Some puppet infrastructure commands refer to this parameter as agent-server-urls, but those commands nonetheless manage the server_list parameter.

| Important: | Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported. |

**provisioned_replicas**

**Purpose**

Specifies the certname of replica to give access to the ca-data file sync repo.

**Node group**

PE HA Master

**Class**

puppet_enterprise::profile::master

**DR-only parameter**

Yes

**Example with enabled replica**

["<REPLICA_CERTNAME>"]

**puppetdb_host**

**Purpose**

Specifies the certname of the node running the PuppetDB service.

**Node group**

PE Master

**Class**

puppet_enterprise::profile::master

**DR-only parameter**

No

**Example with enabled replica**

["<PRIMARY_CERTNAME>","<REPLICA_CERTNAME>"]

**Notes**

Replica values are appended to the end of parameter when a replica is enabled.
Purpose
Specifies the port used for communicating with the PuppetDB service. Always 8081.

Node group
PE Master

Class
puppet_enterprise::profile::master

DR-only parameter
No

Example with enabled replica
[8081,8081]

Notes
Replica values are appended to the end of parameter when a replica is enabled.

replica_hostnames

Purpose
Specifies the certname of the replica to set up pglogical replication for non-PuppetDB databases.

Node group
PE HA Master

Class
puppet_enterprise::profile::database

DR-only parameter
Yes

Example with enabled replica
["<REPLICA_CERTNAME>"]

replicating

Purpose
Specifies whether databases other than PuppetDB replicate data.

Node group
PE Infrastructure

Class
puppet_enterprise

DR-only parameter
Yes

Example with enabled replica
true

Notes
Used when provisioning a new replica.

replication_mode

Purpose
Sets replication type and direction on primary servers and replicas.
Node group

PE Master (none), HA Master (source)

Class

puppet_enterprise::profile::master
puppet_enterprise::profile::database
puppet_enterprise::profile::console

DR-only parameter

Yes (although "none" by default)

Example with enabled replica

PE Master — "none" (Present only in master profile.)

PE HA Master — "source" (Set automatically in the replica profile; no setting in the classifier in PE HA Replica.)

server_list

Purpose

Specifies the list of servers that agents contact, in order.

Node group

PE Agent, PE Infrastructure Agent

Class

puppet_enterprise::profile::agent

DR-only parameter

No

Example with enabled replica

PE Agent — ["<PRIMARY_CERTNAME>:8140","<REPLICA_CERTNAME>:8140"] or in a large installation, ["<LOAD_BALANCER>:8140"]

PE Infrastructure Agent — ["<primary_certname>:8140","<replica_certname>:8140"]

Notes

• Infrastructure nodes must be configured to communicate directly with the primary in the PE Infrastructure Agent node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the PE Agent node group.

• When a replica is enabled, the replica is appended to the end of the list in the PE Infrastructure Agent group, and when not using a load balancer, it’s appended to the list in PE Agent.

• Some puppet infrastructure commands refer to this parameter as agent-server-urls, but those commands nonetheless manage the server_list parameter.

| Important: Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported. |

sync_allowlist

Purpose

Specifies a list of nodes that the primary PuppetDB syncs with.

Node group

PE HA Master
Class
puppet_enterprise::profile::puppetdb

DR-only parameter
Yes

Example with enabled replica
["<REPLICA_CERTNAME>"]

During upgrade, when primary is upgraded but replica hasn't been upgraded, [ ] to prevent syncing until upgrade is complete.

sync_peers

Purpose
Specifies a list of hashes that contain configuration data for syncing with a remote PuppetDB node. Includes the host, port, and sync interval.

Node group
PE HA Master

Class
puppet_enterprise::profile::puppetdb

DR-only parameter
Yes

Example with enabled replica
[{"host":"<REPLICA_CERTNAME>"","port":8081,"sync_interval_minutes":<X}>]

During upgrade, when primary is upgraded but replica hasn't been upgraded, [ ] to prevent syncing until upgrade is complete.

Notes
Updated when you enable a replica.

Load balancer timeout in disaster recovery installations

Disaster recovery configuration uses timeouts to determine when to fail over to the replica. If the load balancer timeout is shorter than the server and agent timeout, connections from agents might be terminated during failover.

To avoid timeouts, set the timeout option for load balancers to four minutes or longer. This duration allows compilers enough time for required queries to PuppetDB and the node classifier service. You can set the load balancer timeout option using parameters in the haproxy or f5 modules.

Configure disaster recovery

To configure disaster recovery, you must provision a replica to serve as backup during failovers. If your primary server is permanently disabled, you can then promote a replica.

Before you begin
Apply disaster recovery system and software requirements.

Tip: The puppet infrastructure commands, which are used to configure and manage disaster recovery, require a valid admin RBAC token and must be run from a root session. Running with elevated privileges via sudo puppet infrastructure is not sufficient. Instead, start a root session by running sudo su -, and then run the puppet infrastructure command. For details about these commands, run puppet infrastructure help <ACTION>, for example, puppet infrastructure help provision.
Related information
Generate a token using puppet-access on page 349
Use puppet-access to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Provision and enable a replica
Provisioning a replica duplicates specific components and services from the primary server to the replica. Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.

Before you begin
- Apply disaster recovery system and software requirements.
- Ensure you have a valid admin RBAC token.
- Ensure Code Manager is enabled and configured on your primary server.
- Move any tuning parameters that you set for your primary server using the console to Hiera. Using Hiera ensures configuration is applied to both your primary server and replica.
- Back up you classifier hierarchy, because enabling a replica alters classification.

Note: While completing this task, the primary server is unavailable to serve catalog requests. Time completing this task accordingly.

1. Configure infrastructure agents to connect orchestration agents to the primary server.
   a) In the console, click Node groups, and in the PE Infrastructure group, select the PE Agent > PE Infrastructure Agent group.
   b) If you manage your load balancers with agents, on the Rules tab, pin load balancers to the group.
      Pinning load balancers to the PE Infrastructure Agent group ensures that they communicate directly with the primary server.
   c) On the Classes tab, find the puppet_enterprise::profile::agent class and specify these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage_puppet_conf</td>
<td>Specify true to ensure that your setting for server_list is configured in the expected location and persists through Puppet runs.</td>
</tr>
<tr>
<td>pcp_broker_list</td>
<td>Hostname for your primary server. Hostnames must include port 8142, for example [&quot;PRIMARY.EXAMPLE.COM:8142&quot;].</td>
</tr>
<tr>
<td>primary_uri server_list</td>
<td>Hostname for your primary server, for example [&quot;PRIMARY.EXAMPLE.COM&quot;]. This setting assumes port 8140 unless you specify otherwise with host:port.</td>
</tr>
</tbody>
</table>
   d) Remove any values set for pcp_broker_ws_uri.
   e) Commit changes.
   f) Run Puppet on all agents classified into the PE Infrastructure Agent group.
2. On the primary server, as the root user, run `puppet infrastructure provision replica <REPLICA NODE NAME> --enable`

**Note:** In installations with compilers, use the `--skip-agent-config` flag with the `--enable` option if you want to:

- Upgrade a replica without needing to run Puppet on all agents.
- Add disaster recovery to an installation without modifying the configuration of existing load balancers.
- Manually configure which load balancer agents communicate with in multi-region installations. See [Managing agent communication in multi-region installations](#) on page 327.

3. Copy your secret key file from the primary server to the replica. The path to the secret key file is `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`.

**Important:** If you do not copy your secret key file onto your replica, the replica generates a new secret key when you promote it. The new key prevents you from accessing credentials for your agentless nodes or running tasks and plans on agentless nodes.

4. Optional: Verify that all services running on the primary server are also running on the replica:
   a) From the primary server, run `puppet infrastructure status --verbose` to verify that the replica is available.
   b) From any managed node, run `puppet agent -t --noop --server_list=<REPLICA HOSTNAME>`. If the replica is correctly configured, the Puppet run succeeds and shows no changed resources.

5. Optional: Deploy updated configuration to agents by running Puppet, or wait for the next scheduled Puppet run.

   If you used the `--skip-agent-config` option, you can skip this step.

**Note:** If you use the direct Puppet workflow, where agents use cached catalogs, you must manually deploy the new configuration by running `puppet job run --no-enforce-environment --query 'nodes {deactivated is null and expired is null}'`

6. Optional: Perform any tests you feel are necessary to verify that Puppet runs continue to work during failover. For example, to simulate an outage on the primary server:
   a) Prevent the replica and a test node from contacting the primary server. For example, you might temporarily shut down the primary server or use `iptables` with drop mode.
   b) Run `puppet agent -t` on the test node. If the replica is correctly configured, the Puppet run succeeds and shows no changed resources. Runs might take longer than normal when in failover mode.
   c) Reconnect the replica and test node.

**Related information**

- [Generate a token using puppet-access](#) on page 349
- [Use puppet-access](#) to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

- [Configure settings with Hiera](#) on page 287
  Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

- [Back up your infrastructure](#) on page 842
  PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

- [Running Puppet on nodes](#) on page 458
Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Managing agent communication in multi-region installations

Typically, when you enable a replica using `puppet infrastructure enable replica`, the configuration tool automatically sets the same communication parameters for all agents. In multi-region installations, with load balancers or compilers in multiple locations, you must manually configure agent communication settings so that agents fail over to the appropriate load balancer or compiler.

To skip automatically configuring which Puppet servers and PCP brokers agents communicate with, use the `--skip-agent-config` flag when you provision and enable a replica, for example:

```
puppet infrastructure provision replica example.puppet.com --enable --skip-agent-config
```

To manually configure which load balancer or compiler agents communicate with, use one of these options:

- **CSR attributes**
  1. For each node, include a CSR attribute that identifies the location of the node, for example `pp_region` or `pp_datacenter`.
  2. Create child groups off of the PE Agent node group for each location.
  3. In each child node group, include the `puppet-enterprise::profile::agent` module and set the `server_list` parameter to the appropriate load balancer or compiler hostname.
  4. In each child node group, add a rule that uses the trusted fact created from the CSR attribute.

- **Hiera**
  For each node or group of nodes, create a key/value pair that sets the `puppet-enterprise::profile::agent::server_list` parameter to be used by the PE Agent node group.

- **Custom method that sets the server_list parameter in puppet.conf**

Promote a replica

If your primary server can’t be restored, you can promote the replica to primary server to establish the replica as the new, permanent primary server.

1. Verify that the primary server is permanently offline.
   
   If the primary server comes back online during promotion, your agents can get confused trying to connect to two active primary servers.

2. On the replica, as the root user, run `puppet infrastructure promote replica`
   
   Promotion can take up to the amount of time it took to install PE initially. Don’t make code or classification changes during or after promotion.

3. When promotion is complete, update any systems or settings that refer to the old primary server, such as PE client tool configurations, Code Manager hooks, and CNAME records.

4. Deploy updated configuration to nodes by running Puppet or waiting for the next scheduled run.

   **Note:** In case of a failover, scheduled Puppet and task runs are rescheduled based on the last execution time.

5. Optional: Provision a new replica in order to maintain disaster recovery.

   **Note:** Agent configuration must be updated before provisioning a new replica. If you re-use your old primary server’s node name for the new replica, agents with outdated configuration might use the new replica as a primary server before it’s fully provisioned.

Related information

**Running Puppet on nodes** on page 458

© 2022 Puppet, Inc., a Perforce company
Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

**Enable a new replica using a failed primary server**

After promoting a replica, you can use your old primary server as a new replica, effectively swapping the roles of your failed primary server and promoted replica.

**Before you begin**

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see Bolt OpenSSH configuration options.

To view all available parameters, use the `--help` flag. The logs for all `puppet infrastructure run` Bolt plans are located at `/var/log/puppetlabs/installer/bolt_info.log`.

You must be able to reach the failed primary server via SSH from the current primary server.

On your promoted replica, as the root user, run `puppet infrastructure run enable_ha_failover`, specifying these parameters:

- **host** — Hostname of the failed primary server. This node becomes your new replica.
- **topology** — Architecture used in your environment, either `mono` (standard) or `mono-with-compile` (large).
- **replication_timeout_secs** — Optional. The number of seconds allowed to complete provisioning and enabling of the new replica before the command fails.
- **tmpdir** — Optional. Path to a directory to use for uploading and executing temporary files.

For example:

```bash
puppet infrastructure run enable_ha_failover host=<FAILED_PRIMARY_HOSTNAME> topology=mono
```

The failed primary server is repurposed as a new replica.

**Forget a replica**

Forgetting a replica cleans up classification and database state, preventing degraded performance over time.

**Before you begin**

Ensure you have a valid admin RBAC token and the replica you want to remove is permanently offline.

Run the `forget` command whenever a replica node is destroyed, even if you plan to replace it with a replica with the same name.

1. On the primary server, as the root user, run `puppet infrastructure forget <REPLICA_NODE_NAME>`
2. Delete your secret key file from the replica because leaving sensitive information on a replica poses a security risk. The path to the secret key file is `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`

The replica is decommissioned, the node is purged as an agent, secret key information is deleted, and a Puppet run is completed on the primary server.

**Related information**

Generate a token using puppet-access on page 349
Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

**Reinitialize a replica**

If you encounter certain errors on your replica after provisioning, you can reinitialize the replica. Reinitializing destroys and re-creates replica databases, as specified.

**Before you begin**

Your primary server must be fully functional and the replica must be able to communicate with the primary server.

⚠️ **CAUTION:** If you reinitialize a functional replica that you already enabled, the replica is unavailable to serve as backup in a failover during reinitialization.

Reinitialization is not intended to fix slow queries or intermittent failures. Reinitialize your replica only if it’s inoperational or you see replication errors.

1. On the replica, as the root user, reinitialize databases as needed:
   - All databases: `puppet infrastructure reinitialize replica`
   - Specific databases: `puppet infrastructure reinitialize replica --db <DATABASE>`
     where `<DATABASE>` is `pe-activity`, `pe-classifier`, `pe-orchestrator`, or `pe-rbac`.
2. Follow prompts to complete the reinitialization.

**Accessing the console**

The console is the web interface for Puppet Enterprise.

Use the console to:

- Manage node requests to join the Puppet deployment.
- Assign Puppet classes to nodes and groups.
- Run Puppet on specific groups of nodes.
- View reports and activity graphs.
- Browse and compare resources on your nodes.
- View package and inventory data.
- Manage console users and their access privileges.

- **Reaching the console** on page 329
  The console is served as a website over SSL, on whichever port you chose when installing the console component.
- **Logging in** on page 330
  Accessing the console requires a username and password.

**Reaching the console**

The console is served as a website over SSL, on whichever port you chose when installing the console component.

Let’s say your console server is `console.domain.com`. If you chose to use the default port (443), you can omit the port from the URL and reach the console by navigating to `https://console.domain.com`.

If you chose to use port 8443, you reach the console at `https://console.domain.com:8443`.

**Remember:** Always use the `https` protocol handler. You cannot reach the console over plain `http`. 

© 2022 Puppet, Inc., a Perforce company
Accepting the console's certificate

The console uses an SSL certificate created by your own local Puppet certificate authority. Because this authority is specific to your site, web browsers won't know it or trust it, and you must add a security exception in order to access the console.

Adding a security exception for the console is safe to do. Your web browser warns you that the console's identity hasn't been verified by one of the external authorities it knows of, but that doesn't mean it's untrustworthy. Because you or another administrator at your site is in full control of which certificates the Puppet certificate authority signs, the authority verifying the site is you.

When your browser warns you that the certificate authority is invalid or unknown:

- In Chrome, click Advanced, then Proceed to <CONSOLE ADDRESS>.
- In Firefox, click Advanced, then Add exception.
- In Internet Explorer or Microsoft Edge, click Continue to this website (not recommended).
- In Safari, click Continue.

Logging in

Accessing the console requires a username and password.

If you are an administrator setting up the console or accessing it for the first time, use the username and password you chose when you installed the console. Otherwise, get credentials from your site's administrator.

Because the console is the main point of control for your infrastructure, it is a good idea to prohibit your browser from storing the login credentials.

Generate a user password reset token

When users forget passwords or lock themselves out of the console by attempting to log in with incorrect credentials too many times, you need to generate a password reset token.

1. In the console, on the Access control page, click the Users tab.
2. Click the name of the user who needs a password reset token.
3. Click Generate password reset. Copy the link provided in the message and send it to the user.

Reset the console administrator password

If you're unable to log in to the console as admin, you can change the password from the command line of the node running console services.

Log in as root to the node running console services (usually your primary server) and reset the console admin password:

```
puppet infrastructure console_password --password=<MY_PASSWORD>
```

Troubleshooting login to the PE admin account

If your directory contains multiple users with a login name of "admin," the PE admin account is unable to log in.

If you are locked out of PE as the admin user and there are no other users with administrator access who you can ask to reset the access control settings in the console, SSH into the box and use curl commands to reset the directory service settings.

For a box named centos7 the curl call looks like this:

```bash
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/rbac-api/v1/ds"
```
Managing access

Role-based access control, more succinctly called RBAC, is used to grant individual users the permission to perform specific actions. Permissions are grouped into user roles, and each user is assigned at least one user role.

By using permissions, you give the appropriate level of access and agency to each user. For example, you can grant users:

- The permission to grant password reset tokens to other users who have forgotten their passwords
- The permission to edit a local user’s metadata
- The permission to deploy Puppet code to specific environments
- The permission to edit class parameters in a node group

You can do access control tasks in the console or using the RBAC API.

- User permissions and user roles on page 331
- Creating and managing local users and user roles on page 338
- Connecting external directory services to PE on page 340

Puppet Enterprise’s role-based access control (RBAC) enables you to manage users—what they can create, edit, or view, and what they can’t—in an organized, high-level way that is vastly more efficient than managing user permissions on a per-user basis. User roles are sets of permissions you can apply to multiple users. You can’t assign permissions to single users in PE, only to user roles.

- Working with user groups from an external directory service on page 346
- Token-based authentication on page 347

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

- RBAC API on page 353
- Activity service API on page 382

The activity service logs changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles.

User permissions and user roles

The "role" in role-based access control refers to a system of user roles, which are assigned to user groups and their users. Those roles contain permissions, which define what a user can or can't do within PE.

When you add new users to PE, they can't actually do anything until they're associated with a user role, either explicitly through role assignment or implicitly through group membership and role inheritance. When a user is added to a role, they receive all the permissions of that role.
There are five default user roles:

**Administrators**
Manage users and permissions, create and modify node groups and other objects. Administrators have all permissions assigned to them by default.

**Operators**
Create and modify node groups and other objects.

**Viewers**
View, but can't modify, objects in the console.

**Code Deployers**
Synchronize code from version control systems to Puppet Server.

**Project Deployers**
Deploy projects and run project tasks and plans.

You can also create custom roles. For example, you might want to create a user role that grants users permission to view but not edit a specific subset of node groups. Or you might want to divide up administrative privileges so that one user role is able to reset passwords while another can edit roles and create users.

Permissions are additive. If a user is associated with multiple roles, that user is able to perform all of the actions described by all of the permissions on all of the applied roles.

**Structure of user permissions**

User permissions are structured as a triple of type, permission, and object.

- **Types** are everything that can be acted on, such as node groups, users, or user roles.
- **Permissions** are what you can do with each type, such as create, edit, or view.
- **Objects** are specific instances of the type.

Some permissions added to the Administrators user role might look like this:

<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node groups</td>
<td>View</td>
<td>PE Master</td>
<td>Gives permission to view the PE Master node group.</td>
</tr>
<tr>
<td>User roles</td>
<td>Edit</td>
<td>All</td>
<td>Gives permission to edit all user roles.</td>
</tr>
</tbody>
</table>

When no object is specified, a permission applies to all objects of that type. In those cases, the object is “All”. This is denoted by "*" in the API.

In both the console and the API, "*" is used to express a permission for which an object doesn’t make sense, such as when creating users.

**User permissions and default roles**

The table below lists the available permissions, an explanation of the permission and how to use it, and which role has that permission assigned to it by default, if any.

| Note | The Administrator role has all permissions assigned to it, so to avoid redundancy it isn’t listed in the table. |

Types and permissions have both a display name, which is the name you see in the console interface, and a system name, which is the name you use to construct permissions in the API. In the following table, the display names are used.
<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Definition</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate request</td>
<td>Accept and reject</td>
<td>Accept and reject certificate signing requests. Object must always be &quot;*&quot;.</td>
<td>Operators</td>
</tr>
<tr>
<td>Console</td>
<td>View</td>
<td>View the PE console. Object must always be &quot;*&quot;.</td>
<td>Operators, Viewers</td>
</tr>
<tr>
<td>Directory service</td>
<td>View, edit, and test</td>
<td>View, edit, and test directory service settings. Object must always be &quot;*&quot;.</td>
<td>Operators</td>
</tr>
<tr>
<td>Job orchestrator</td>
<td>Start, stop and view jobs</td>
<td>Start and stop jobs and tasks, view jobs and job progress, and view an inventory of nodes that are connected to the PCP broker.</td>
<td>Operators, Viewers, Project Deployers</td>
</tr>
<tr>
<td>Node groups</td>
<td>Create, edit, and delete child groups</td>
<td>Create new child groups, delete existing child groups, and modify every attribute of child groups except environment. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Node groups</td>
<td>Edit child group rules</td>
<td>Edit the rules of descendents of a node group. This does not grant the ability to edit the rules of the group in the object field, only children of that group. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Node groups</td>
<td>Edit classes, parameters, and variables</td>
<td>Edit every attribute of a node group except its environment and rule. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Node groups</td>
<td>Edit configuration data</td>
<td>Edit parameterized configuration data on a node group. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Type</td>
<td>Permission</td>
<td>Definition</td>
<td>Roles</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Node groups</td>
<td>Edit parameters and variables</td>
<td>Edit the class parameters and variables of a node group's classes. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Node groups</td>
<td>Set environment</td>
<td>Set the environment of a node group. This permission is inherited by all descendents of the node group.</td>
<td>Operators</td>
</tr>
<tr>
<td>Node groups</td>
<td>View</td>
<td>See all attributes of a node group, most notably the values of class parameters and variables. This permission is inherited by all descendents of the node group.</td>
<td>• Operators</td>
</tr>
<tr>
<td>• Viewers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Edit node data from PuppetDB</td>
<td>Edit node data imported from PuppetDB. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>View node data from PuppetDB</td>
<td>View node data imported from PuppetDB. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>View sensitive connection information in inventory service</td>
<td>View sensitive parameters stored in the inventory service for a connection. For example, user credentials. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>Add and delete connection information from inventory service</td>
<td>Add new connections to the inventory service and delete existing connections.</td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>Run plans</td>
<td>Run specific plans on all nodes.</td>
<td></td>
</tr>
<tr>
<td>Projects</td>
<td>Deploy projects</td>
<td>Not used.</td>
<td>Project Deployers</td>
</tr>
<tr>
<td>Projects</td>
<td>Run tasks and plans from projects</td>
<td>Not used.</td>
<td>Project Deployers</td>
</tr>
<tr>
<td>Puppet agent</td>
<td>Run Puppet on agent nodes</td>
<td>Trigger a Puppet run from the console or orchestrator. Object must always be &quot;*&quot;.</td>
<td>Operators</td>
</tr>
<tr>
<td>Puppet environment</td>
<td>Deploy code</td>
<td>Deploy code to a specific PE environment.</td>
<td>• Operators</td>
</tr>
<tr>
<td>Type</td>
<td>Permission</td>
<td>Definition</td>
<td>Roles</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>Compile catalogs for remote nodes</td>
<td>Compile a catalog for any node managed by this PE instance. This permission is required to run impact analysis tasks in Continuous Delivery for Puppet Enterprise.</td>
<td></td>
</tr>
<tr>
<td>Scheduled jobs</td>
<td>Delete another user's scheduled jobs</td>
<td>Delete scheduled jobs created by the user instance specified in the permission. This can be granted per user.</td>
<td></td>
</tr>
<tr>
<td>Tasks</td>
<td>Run tasks</td>
<td>Run specific tasks on all nodes, a selected node group, or nodes that match a PQL query.</td>
<td></td>
</tr>
<tr>
<td>Important: A task must be permitted to run on all nodes in order to run on nodes that are outside of the PuppetDB (over SSH or WinRM for example). As a result, users with such permissions can run tasks on any nodes they have the credentials to access.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User groups</td>
<td>Delete</td>
<td>Delete a user group. This can be granted per group.</td>
<td></td>
</tr>
<tr>
<td>User groups</td>
<td>Import</td>
<td>Import groups from the directory service for use in RBAC. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>User roles</td>
<td>Create</td>
<td>Create new roles. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>User roles</td>
<td>Edit</td>
<td>Edit and delete a role. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>User roles</td>
<td>Edit members</td>
<td>Change which users and groups a role is assigned to. This can be granted per role.</td>
<td></td>
</tr>
<tr>
<td>Users</td>
<td>Create</td>
<td>Create new local users. Remote users are &quot;created&quot; by that user authenticating for the first time with RBAC. Object must always be &quot;*&quot;.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Permission</td>
<td>Definition</td>
<td>Roles</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Users</td>
<td>Edit</td>
<td>Edit a local user's data, such as name or email, and delete a local or remote user from PE. This can be granted per user.</td>
<td></td>
</tr>
<tr>
<td>Users</td>
<td>Reset password</td>
<td>Grant password reset tokens to users who have forgotten their passwords. This process also reinstates a user after the use has been revoked. This can be granted per user.</td>
<td></td>
</tr>
<tr>
<td>Users</td>
<td>Revoke</td>
<td>Revoke or disable a user. This means the user is no longer able to authenticate and use the console, node classifier, or RBAC. This permission also includes the ability to revoke the user's authentication tokens. This can be granted per user.</td>
<td></td>
</tr>
</tbody>
</table>

**Display names and corresponding system names**

The following table provides both the display and system names for the types and all their corresponding permissions.

<table>
<thead>
<tr>
<th>Type (display name)</th>
<th>Type (system name)</th>
<th>Permission (display name)</th>
<th>Permission (system name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate requests</td>
<td>cert_requests</td>
<td>Accept and reject</td>
<td>accept_reject</td>
</tr>
<tr>
<td>Console</td>
<td>console_page</td>
<td>View</td>
<td>view</td>
</tr>
<tr>
<td>Directory service</td>
<td>directory_service</td>
<td>View, edit, and test</td>
<td>edit</td>
</tr>
<tr>
<td>Job orchestrator</td>
<td>orchestrator</td>
<td>Start, stop and view jobs</td>
<td>view</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Create, edit, and delete child groups</td>
<td>modify_children</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Edit child group rules</td>
<td>edit_child_rules</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Edit classes, parameters, and variables</td>
<td>edit_classification</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Edit configuration data</td>
<td>edit_config_data</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Edit parameters and variables</td>
<td>edit_params_and_vars</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>Set environment</td>
<td>set_environment</td>
</tr>
<tr>
<td>Node groups</td>
<td>node_groups</td>
<td>View</td>
<td>view</td>
</tr>
<tr>
<td>Nodes</td>
<td>nodes</td>
<td>Edit node data from PuppetDB</td>
<td>edit_data</td>
</tr>
<tr>
<td>Nodes</td>
<td>nodes</td>
<td>View node data from PuppetDB</td>
<td>view_data</td>
</tr>
<tr>
<td>Type (display name)</td>
<td>Type (system name)</td>
<td>Permission (display name)</td>
<td>Permission (system name)</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Nodes</td>
<td>nodes</td>
<td>View sensitive connection information in inventory service</td>
<td>view_inventory_sensitive</td>
</tr>
<tr>
<td>Plans</td>
<td>plans</td>
<td>Run Plans</td>
<td>run</td>
</tr>
<tr>
<td>Puppet agent</td>
<td>puppet_agent</td>
<td>Run Puppet on agent nodes</td>
<td>run</td>
</tr>
<tr>
<td>Puppet environment</td>
<td>environment</td>
<td>Deploy code</td>
<td>deploy_code</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>puppetserver</td>
<td>Compile catalogs for remote nodes</td>
<td>compile_catalogs</td>
</tr>
<tr>
<td>Tasks</td>
<td>tasks</td>
<td>Run Tasks</td>
<td>run</td>
</tr>
<tr>
<td>User groups</td>
<td>user_groups</td>
<td>Import</td>
<td>import</td>
</tr>
<tr>
<td>User roles</td>
<td>user_roles</td>
<td>Create</td>
<td>create</td>
</tr>
<tr>
<td>User roles</td>
<td>user_roles</td>
<td>Edit</td>
<td>edit</td>
</tr>
<tr>
<td>User roles</td>
<td>user_roles</td>
<td>Edit members</td>
<td>edit_members</td>
</tr>
<tr>
<td>Users</td>
<td>users</td>
<td>Create</td>
<td>create</td>
</tr>
<tr>
<td>Users</td>
<td>users</td>
<td>Edit</td>
<td>edit</td>
</tr>
<tr>
<td>Users</td>
<td>users</td>
<td>Reset password</td>
<td>reset_password</td>
</tr>
<tr>
<td>Users</td>
<td>users</td>
<td>Revoke</td>
<td>disable</td>
</tr>
</tbody>
</table>

**Related information**

Permissions endpoints on page 369

You assign permissions to user roles to manage user access to objects. The permissions endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

**Working with node group permissions**

Node groups in the node classifier are structured hierarchically; therefore, node group permissions inherit. Users with specific permissions on a node group implicitly receive the permissions on any child groups below that node group in the hierarchy.

Two types of permissions affect a node group: those that affect a group itself, and those that affect the group's child groups. For example, giving a user the "Set environment" permission on a group allows the user to set the environment for that group and all of its children. On the other hand, assigning "Edit child group rules" to a group allows a user to edit the rules for any child group of a specified node group, but not for the node group itself. This allows some users to edit aspects of a group, while other users can be given permissions for all children of that group without being able to affect the group itself.

Due to the hierarchical nature of node groups, if a user is given a permission on the default (All) node group, this is functionally equivalent to giving them that permission on "* * *".

**Best practices for assigning permissions**

Working with user permissions can be a little tricky. You don't want to grant users permissions that essentially escalate their role, for example. The following sections describe some strategies and requirements for setting permissions.

**Grant edit permissions to users with create permissions**

Creating new objects doesn't automatically grant the creator permission to view those objects. Therefore, users who have permission to create roles, for example, must also be given permission to edit roles, or they won't be able to see...
new roles that they create. Our recommendation is to assign users permission to edit all objects of the type that they have permission to create. For example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>User roles</td>
<td>Edit members</td>
<td>All (or &quot;<em>&quot;</em>)</td>
</tr>
<tr>
<td>Users</td>
<td>Edit</td>
<td>All (or &quot;<em>&quot;</em>)</td>
</tr>
</tbody>
</table>

**Avoid granting overly permissive permissions**

Operators, a default role in PE, have many of the same permissions as Administrators. However, we've intentionally limited this role's ability to edit user roles. This way, members of this group can do many of the same things as Administrators, but they can't edit (or enhance) their own permissions.

Similarly, avoid granting users more permissions than their roles allow. For example, if users have the `roles:edit:*` permission, they are able to add the `node_groups:view:*` permission to the roles they belong to, and subsequently see all node groups.

**Give permission to edit directory service settings to the appropriate users**

The directory service password is not redacted when the settings are requested in the API. Give `directory_service:edit:*` permission only to users who are allowed see the password and other settings.

**The ability to reset passwords should be given only with other password permissions**

The ability to help reset passwords for users who forgot them is granted by the `users:reset_password:<instance>` permission. This permission has the side effect of reinstating revoked users after the reset token is used. As such, the reset password permission should be given only to users who are also allowed to revoke and reinstate other users.

**Creating and managing local users and user roles**

Puppet Enterprise's role-based access control (RBAC) enables you to manage users—what they can create, edit, or view, and what they can't—in an organized, high-level way that is vastly more efficient than managing user permissions on a per-user basis. User roles are sets of permissions you can apply to multiple users. You can't assign permissions to single users in PE, only to user roles.

In addition to creating users, there are two default users that come with PE: Administrator and API User. The Administrator is a user that has the Administrator role applied by default, meaning they have every permission. The Administrator user is revokable in case users are managed through a directory service, like LDAP.

The API user is for service-to-service authentication within PE. It cannot be used with the standard login, cannot be revoked, and it is only available through certificate-based authentication. The RBAC allow list identifies (by certname) the certificates you can use for API user authentication.

**Remember:** Each user must be assigned to one or more roles before they can log in and use PE.

**Note:** Puppet stores local accounts and directory service integration credentials securely. Local account passwords are hashed using SHA-256 multiple times along with a 32-bit salt. Directory service lookup credentials configured for directory lookup purposes are encrypted using AES-128. Puppet does not store the directory credentials used for authenticating to Puppet. These are different from the directory service lookup credentials.

**Create a new user**

These steps add a local user.

To add users from an external directory, see Working with user groups from an external directory.
1. In the console, on the **Access control** page, click the **Users** tab.
2. In the **Full name** field, enter the user's full name.
3. In the **Login** field, enter a username for the user.
4. Click **Add local user**.

**Give a new user access to the console**

When you create new local users, you need to send them a password reset token so that they can log in for the first time.

1. On the **Access control** page, on the **Users** tab, select the user's full name.
2. Click **Generate password reset**.
3. Copy the link provided in the message and send it to the new user.

**Create a new user role**

RBAC has four predefined roles: Administrators, Code Deployers, Operators, and Viewers. You can also define your own custom user roles.

Users with the appropriate permissions, such as Administrators, can define custom roles. To avoid potential privilege escalation, only users who are allowed all permissions should be given the permission to edit user roles.

1. In the console, on the **Access control** page, click the **User roles** tab.
2. In the **Name** field, enter a name for the new user role.
3. (Optional) In the **Description** field, enter a description of the new user role.
4. Click **Add role**.

**Assign permissions to a user role**

You can mix and match permissions to create custom user roles that provide users with precise levels of access to PE actions.

**Before you begin**

Review User permissions and user roles, which includes important information about how permissions work in PE.

1. On the **Access control** page, on the **User roles** tab, select a user role.
2. Click **Permissions**.
3. In the **Type** field, select the type of object you want to assign permissions for, such as **Node groups**.
4. In the **Permission** field, select the permission you want to assign, such as **View**.
5. In the **Object** field, select the specific object you want to assign the permission to. For example, if you are setting a permission to view node groups, select a specific node group this user role has permissions to view.
6. Click **Add permission**, and commit changes.

**Related information**

Best practices for assigning permissions on page 337

Working with user permissions can be a little tricky. You don't want to grant users permissions that essentially escalate their role, for example. The following sections describe some strategies and requirements for setting permissions.

**Add a user to a user role**

When you add users to a role, the user gains the permissions that are applied to that role. A user can't do anything in PE until they have been assigned to a role.

1. On the **Access control** page, on the **User roles** tab, select a user role.
2. Click **Member users**.
3. In the **User name** field, select the user you want to add to the user role.
4. Click Add user, and commit changes.

Remove a user from a user role
You can change a user's permissions by removing them from a user role. The user loses the permissions associated with the role, and won't be able to do anything in PE until they are assigned to a new role.

1. On the Access control page, on the User roles tab, select a user role.
2. Click Member users.
3. Locate the user you want to remove from the user role. Click Remove, and commit changes.

Revoke a user's access
If you want to remove a user's access to PE but not delete their account, you can revoke them. Revocation is also what happens when a user is locked out from too many incorrect password attempts.

1. In the console, on the Access control page, click the Users tab.
2. In the Full name column, select the user you want to revoke.
3. Click Revoke user access.

Tip: To unrevoke a user, follow the steps above and click Reinststate user access.

Delete a user
You can delete a user through the console. Note, however, that this action deletes only the user's Puppet Enterprise account, not the user's listing in any external directory service.

Deletion removes all data about the user except for their activity data, which continues to be stored in the database and remains viewable through the API.

1. In the console, on the Access control page, click the Users tab.
2. In the Full name column, locate the user you want to delete.
3. Click Remove.

Note: Users with superuser privileges cannot be deleted, and the Remove button does not appear for these users.

CAUTION: If a user is deleted from the console and then recreated with the same full name and login, PE issues the recreated user a new unique user ID. In this instance, queries for the login to the API's activity database return information on both the deleted user and the new user. However, in the console, the new user's Activity tab does not display information about the deleted user's account.

Delete a user role
You can delete a user role through the console.

When you delete a user role, users lose the permissions that the role gives them. This can impact their access to Puppet Enterprise if they have not been assigned other user roles.

1. In the console, on the Access control page, click the User roles tab.
2. In the Name column, locate the role you want to delete.
3. Click Remove.

Connecting external directory services to PE
Puppet Enterprise connects to external Lightweight Directory Access Protocol (LDAP) directory services through its role-based access control (RBAC) service. Because PE integrates with cloud LDAP service providers such as Okta, you can use existing users and user groups that have been set up in your external directory service.
Specifically, you can:

- Authenticate external directory users.
- Authorize access of external directory users based on RBAC permissions.
- Store and retrieve the groups and group membership information that has been set up in your external directory.

**Note:** Puppet stores local accounts and directory service integration credentials securely. Local account passwords are hashed using SHA-256 multiple times along with a 32-bit salt. Directory service lookup credentials configured for directory lookup purposes are encrypted using AES-128. Puppet does not store the directory credentials used for authenticating to Puppet. These are different from the directory service lookup credentials.

PE supports OpenLDAP and Active Directory. If you have predefined groups in OpenLDAP or Active Directory, you can import these groups into the console and assign user roles to them. Users in an imported group inherit the permissions specified in assigned user roles. If new users are added to the group in the external directory, they also inherit the permissions of the role to which that group belongs.

**Note:** The connection to OpenLDAP and Active Directory is read-only. If you want to make changes to remote users or user groups, you need to edit the information directly in the external directory.

### Connect to an external directory service

PE connects to the external directory service when a user logs in or when groups are imported. The supported directory services are OpenLDAP and Active Directory.

1. In the console, on the **Access control** page, click the **External directory** tab.
2. Fill in the directory information.
   
   All fields are required, except for **Login help**, **Lookup user**, **Lookup password**, **User relative distinguished name**, and **Group relative distinguished name**.

   If you do not enter **User relative distinguished name** or **Group relative distinguished name**, RBAC searches the entire base DN for the user or group.

3. Click **Test connection** to ensure that the connection has been established. Save your settings after you have successfully tested them.

   **Note:** This only tests the connection to the LDAP server. It does not test or validate LDAP queries.

### External directory settings

The table below provides examples of the settings used to connect to an Active Directory service and an OpenLDAP service to PE. Each setting is explained in more detail below the table.

**Important:** The settings shown in the table are examples. You need to substitute these example settings with the settings used in your directory service.

<table>
<thead>
<tr>
<th>Name</th>
<th>Example Active Directory settings</th>
<th>Example OpenLDAP settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory name</td>
<td>My Active Directory</td>
<td>My Open LDAP Directory</td>
</tr>
<tr>
<td>Hostname</td>
<td>myhost.delivery.exampleservice.net</td>
<td>myhost.delivery.exampleservice.net</td>
</tr>
<tr>
<td>Port</td>
<td>389 (636 for LDAPS)</td>
<td>389 (636 for LDAPS)</td>
</tr>
<tr>
<td>Lookup user (optional)</td>
<td>cn=queryuser, cn=Users, dc=puppetlabs, dc=admin, dc=delivery, dc=puppetlabs, dc=net</td>
<td></td>
</tr>
<tr>
<td>Lookup password (optional)</td>
<td>The lookup user's password.</td>
<td>The lookup user's password.</td>
</tr>
<tr>
<td>Connection timeout (seconds)</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Name</td>
<td>Example Active Directory settings</td>
<td>Example OpenLDAP settings</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Connect using:</td>
<td>SSL</td>
<td>StartTLS</td>
</tr>
<tr>
<td>Validate the hostname?</td>
<td>Default is yes.</td>
<td>Default is yes.</td>
</tr>
<tr>
<td>Allow wildcards in SSL certificate?</td>
<td>Default is no.</td>
<td>Default is no.</td>
</tr>
<tr>
<td>Base distinguished name</td>
<td>dc=puppetlabs,dc=com</td>
<td>dc=puppetlabs,dc=com</td>
</tr>
<tr>
<td>User login attribute</td>
<td>sAMAccountName</td>
<td>cn</td>
</tr>
<tr>
<td>User email address</td>
<td>mail</td>
<td>mail</td>
</tr>
<tr>
<td>User full name</td>
<td>displayName</td>
<td>displayName</td>
</tr>
<tr>
<td>User relative distinguished name (optional)</td>
<td>cn=users</td>
<td>ou=users</td>
</tr>
<tr>
<td>Group object class</td>
<td>group</td>
<td>groupOfUniqueNames</td>
</tr>
<tr>
<td>Group membership field</td>
<td>member</td>
<td>uniqueMember</td>
</tr>
<tr>
<td>Group name attribute</td>
<td>name</td>
<td>displayName</td>
</tr>
<tr>
<td>Group lookup attribute</td>
<td>cn</td>
<td>cn</td>
</tr>
<tr>
<td>Group relative distinguished name (optional)</td>
<td>cn=groups</td>
<td>ou=groups</td>
</tr>
<tr>
<td>Turn off LDAP_MATCHING_RULE_IN_CHAIN?</td>
<td>Default is no.</td>
<td>Default is no.</td>
</tr>
<tr>
<td>Search nested groups?</td>
<td>Default is no.</td>
<td>Default is no.</td>
</tr>
</tbody>
</table>

**Explanation of external directory settings**

**Directory name** The name that you provide here is used to refer to the external directory service anywhere it is used in the PE console. For example, when you view a remote user in the console, the name that you provide in this field is listed in the console as the source for that user. Set any name of your choice.

**Login help (optional)** If you supply a URL here, a "Need help logging in?" link is displayed on the login screen. The href attribute of this link is set to the URL that you provide.

**Hostname** The FQDN of the directory service to which you are connecting.

**Port** The port that PE uses to access the directory service. The port is generally 389, unless you choose to connect using SSL, in which case it is generally 636.

**Lookup user (optional)** The distinguished name (DN) of the directory service user account that PE uses to query information about users and groups in the directory server. If a username is supplied, this user must have read access for all directory entries that are to be used in the console. We recommend that this user is restricted to read-only access to the directory service.

If your LDAP server is configured to allow anonymous binding, you do not need to provide a lookup user. In this case, the RBAC service binds anonymously to your LDAP server.

**Lookup password (optional)** The lookup user's password.

If your LDAP server is configured to allow anonymous binding, you do not need to provide a lookup password. In this case, the RBAC service binds anonymously to your LDAP server.

**Connection timeout (seconds)** The number of seconds that PE attempts to connect to the directory server before timing out. Ten seconds is fine in the majority of cases. If you are experiencing timeout errors, make sure the directory service is up and reachable, and then increase the timeout if necessary.
**Connect using:** Select the security protocol you want to use to connect to the external directory: **SSL** and **StartTLS** encrypt the data transmitted. **Plain text** is not a secure connection. In addition, to ensure that the directory service is properly identified, configure the `ds-trust-chain` to point to a copy of the public key for the directory service. For more information, see **Verify directory server certificates** on page 345.

**Validate the hostname?** Select **Yes** to verify that the Directory Services hostname used to connect to the LDAP server matches the hostname on the SSL certificate. This option is not available when you choose to connect to the external directory using plain text.

**Allow wildcards in SSL certificate?** Select **Yes** to allow a connection to a Directory Services server with a SSL certificates that use a wildcard (*) specification. This option is not available when you choose to connect to the external directory using plain text.

**Base distinguished name** When PE constructs queries to your external directory (for example to look up user groups or users), the queries consist of the relative distinguished name (RDN) (optional) + the base distinguished name (DN), and are then filtered by lookup/login attributes. For example, if PE wants to authenticate a user named Bob who has the RDN `ou=bob,ou=users`, it sends a query in which the RDN is concatenated with the DN specified in this field (for example, `dc=puppetlabs,dc=com`). This gives a search base of `ou=bob,ou=users,dc=puppetlabs,dc=com`.

The base DN that you provide in this field specifies where in the directory service tree to search for groups and users. It is the part of the DN that all users and groups that you want to use have in common. It is commonly the root DN (example `dc=example,dc=com`) but in the following example of a directory service entry, you could set the base DN to `ou=Puppet,dc=example,dc=com` because both the group and the user are also under the organizational unit `ou=Puppet`.

**Example directory service entry**

```bash
# A user named Harold
dn: cn=harold,ou=Users,ou=Puppet,dc=example,dc=com
objectClass: organizationalPerson
cn: harold
displayName: Harold J.
mail: harold@example.com
memberOf: inspectors
sAMAccountName: harold11

# A group Harold is in
dn: cn=inspectors,ou=Groups,ou=Puppet,dc=example,dc=com
objectClass: group
cn: inspectors
displayName: The Inspectors
member: harold
```

**User login attribute** This is the directory attribute that the user uses to log in to PE. For example, if you specify `sAMAccountName` as the user login attribute, Harold logs in with the username "harold11" because `sAMAccountName=harold11` in the example directory service entry provided above.

The value provided by the user login attribute must be unique among all entries under the User RDN + Base DN search base you’ve set up.

For example, say you’ve selected the following settings:

```bash
base DN = dc=example,dc=com
user RDN = null
user login attribute = cn
```

When Harold tries to log in, the console searches the external directory for any entries under `dc=example,dc=com` that have the attribute/value pair `cn=harold`. (This attribute/value pair does not need to be contained within the DN). However, if there is another user named Harold who has the DN `cn=harold,ou=OtherUsers,dc=example,dc=com`, two results are returned and the login does not succeed because the console does not know which entry to use. Resolve this issue by either narrowing your search base such
that only one of the entries can be found, or using a value for login attribute that you know to be unique. This makes sAMAccountName a good choice if you’re using Active Directory, as it must be unique across the entire directory.

**User email address** The directory attribute to use when displaying the user's email address in PE.

**User full name** The directory attribute to use when displaying the user's full name in PE.

**User relative distinguished name (optional)** The user RDN that you set here is concatenated with the base DN to form the search base when looking up a user. For example, if you specify ou=users for the user RDN, and your base DN setting is ou=Puppet,dc=example,dc=com, PE finds users that have ou=users,ou=Puppet,dc=example,dc=com in their DN.

This setting is optional. If you choose not to set it, PE searches for the user in the base DN (example: ou=Puppet,dc=example,dc=com). Setting a user RDN is helpful in the following situations:

- When you experience long wait times for operations that contact the directory service (either when logging in or importing a group for the first time). Specifying a user RDN reduces the number of entries that are searched.
- When you have more than one entry under your base DN with the same login value.

**Tip:** It is not currently possible to specify multiple user RDNs. If you want to filter RDNs when constructing your query, we suggest creating a new lookup user who only has read access for the users and groups you want to use in PE.

**Group object class** The name of an object class that all groups have.

**Group membership field** Tells PE how to find which users belong to which groups. This is the name of the attribute in the external directory groups that indicates who the group members are.

**Group name attribute** The attribute that stores the display name for groups. This is used for display purposes only.

**Group lookup attribute** The value used to import groups into PE. Given the example directory service entry provided above, the group lookup attribute would be cn. When specifying the Inspectors group in the console to import it, provide the name inspectors.

The value for this attribute must be unique under your search base. If you have users with the same login as the lookup of a group that you want to use, you can narrow the search base, use a value for the lookup attribute that you know to be unique, or specify the **Group object class** that all of your groups have in common but your users do not.

**Tip:** If you have a large number of nested groups in your group hierarchy, or you experience slowness when logging in with RBAC, we recommend disabling nested group search unless you need it for your authorization schema to work.

**Group relative distinguished name (optional)** The group RDN that you set here is concatenated with the base DN to form the search base when looking up a group. For example, if you specify ou=groups for the group RDN, and your base DN setting is ou=Puppet,dc=example,dc=com, PE finds groups that have ou=groups,ou=Puppet,dc=example,dc=com in their DN.

This setting is optional. If you choose not to set it, PE searches for the group in the base DN (example: ou=Puppet,dc=example,dc=com). Setting a group RDN is helpful in the following situations:

- When you experience long wait times for operations that contact the directory service (either when logging in or importing a group for the first time). Specifying a group RDN reduces the number of entries that are searched.
- When you have more than one entry under your base DN with the same lookup value.

**Tip:** It is not currently possible to specify multiple group RDNs. If you want to filter RDNs when constructing your query, create a new lookup user who only has read access for the users and groups you plan to use in PE.

**Note:** At present, PE supports only a single Base DN. Use of multiple user RDNs or group RDNs is not supported.
**Turn off LDAP_MATCHING_RULE_IN_CHAIN?** Select Yes to turn off the LDAP matching rule that looks up the chain of ancestry for an object until it finds a match. For organizations with a large number of group memberships, matching rule in chain can slow performance.

**Search nested groups?** Select Yes to search for groups that are members of an external directory group. For organizations with a large number of nested group memberships, searching nested groups can slow performance.

**Related information**

PUT /ds on page 375
Replaces current directory service connection settings. Authentication is required.

---

**Verify directory server certificates**

To ensure that RBAC isn't being subjected to a Man-in-the Middle (MITM) attack, verify the directory server's certificate.

When you select SSL or StartTLS as the security protocol to use for communications between PE and your directory server, the connection to the directory is encrypted. To ensure that the directory service is properly identified, configure the `ds-trust-chain` to point to a copy of the public key for the directory service.

The RBAC service verifies directory server certificates using a trust store file, in Java Key Store (JKS), PEM, or PKCS12 format, that contains the chain of trust for the directory server's certificate. This file needs to exist on disk in a location that is readable by the user running the RBAC service.

To turn on verification:

1. In the console, click **Node groups**.
2. Open the **PE Infrastructure** node group and select the **PE Console** node group.
3. Click **Classes**. Locate the `puppet_enterprise::profile::console` class.
4. In the **Parameter** field, select `rbac_ds_trust_chain`.
5. In the **Value** field, set the absolute path to the trust store file.
6. Click **Add parameter**, and commit changes.
7. To make the change take effect, run Puppet. Running Puppet restarts pe-console-services.

After this value is set, the directory server's certificate is verified whenever RBAC is configured to connect to the directory server using SSL or StartTLS.

---

**Enable custom password policies through LDAP**

Password policies are not configurable in PE. However, if your organization requires more complex password policies than the default, you can allow LDAP to manage custom password policies by delegating administrative privileges to LDAP and then revoking the admin user in the console.

**Before you begin**

Enable LDAP by [Connecting external directory services to PE](#) on page 340. Make sure you are logged into LDAP as the administrative user and have given LDAP administrative privileges.

Revoke the admin user:

1. In the console, on the **Access control** page, click the **Users** tab.
2. Select **Administrator**.
3. On the **User details** page, click **Revoke user access**.

You have revoked the admin user in the console, which allows LDAP to manage password policies. To enable the admin again, select **Reinstate user access** on the admin's **User details** page.
Working with user groups from an external directory service

You don’t explicitly add remote users to PE. Instead, after the external directory service has been successfully connected, remote users must log into PE, which creates their user record.

If the user belongs to an external directory group that has been imported into PE and then assigned to a role, the user is assigned to that role and gains the privileges of the role. Roles are additive: You can assign users to more than one role, and they gain the privileges of all the roles to which they are assigned.

Import a user group from an external directory service

You import existing external directory groups to PE explicitly, which means you add the group by name.

1. In the console, on the Access control page, click the User groups tab.

   User groups is available only if you have established a connection with an external directory.

2. In the Login field, enter the name of a group from your external directory.

3. Click Add group.

   Remember: No user roles are listed until you add this group to a role. No users are listed until a user who belongs to this group logs into PE.

Troubleshooting: A PE user and user group have the same name

If you have both a PE user and an external directory user group with the exact same name, PE throws an error when you try to log on as that user or import the user group.

To work around this problem, you can change your settings to use different RDNs for users and groups. This works as long as all of your users are contained under one RDN that is unique from the RDN that contains all of your groups.

Assign a user group to a user role

After you’ve imported a group, you can assign it a user role, which gives each group member the permissions associated with that role. You can add user groups to existing roles, or you can create a new role, and then add the group to the new role.

1. In the console, on the Access control page, click the User roles tab.

2. Click the role you want to add the user group to.

3. Click Member groups. In the Group name field, select the user group you want to add to the user role.

4. Click Add group, and commit changes.

Delete a user group

You can delete a user group in the console. Users who were part of the deleted group lose the permissions associated with roles assigned to the group.

   Remember: This action removes the group only from Puppet Enterprise, not from the associated external directory service.

1. In the console, on the Access control page, click the User groups tab.

   User groups is available only if you have established a connection with an external directory.

2. Locate the group that you wish to delete.

3. Click Remove.
Removing a remote user’s access to PE

In order to fully revoke the remote user’s access to Puppet Enterprise, you must also remove the user from the external directory groups accessed by PE.

Deleting a remote user’s PE account does not automatically prevent that user from accessing PE in the future. So long as the remote user is still a member of a group in an external directory that PE is configured to access, the user retains the ability to log into PE.

If you delete a user from your external directory service but not from PE, the user can no longer log in, but any generated tokens or existing console sessions continue to be valid until they expire. To invalidate the user's tokens or sessions, revoke the user's PE account, which also automatically revokes all tokens for the user. You must manually delete the user for their account record to disappear.

Token-based authentication

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

You can generate authentication tokens using puppet-access or the RBAC API V2 endpoint. You can also generate one-off tokens that do not need to be saved, which are typically used by a service.

RBAC settings for authentication tokens can be managed in the console in the PE Infrastructure group. See Configure RBAC and token-based authentication settings on page 295 for a list of settings you can change.

Authentication tokens manage access to the following PE services:

- Puppet orchestrator
- Code Manager
- Node classifier
- Role-based access control (RBAC)
- Activity service
- PuppetDB

Related information

Installing client tools on page 250
PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

User permissions and default roles on page 332
The table below lists the available permissions, an explanation of the permission and how to use it, and which role has that permission assigned to it by default, if any.

Configure puppet-access

The puppet-access command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. Ensure puppet-access is configured correctly before using it to generate authentication tokens.

The configuration file for puppet-access allows you to generate tokens from the CLI without having to pass additional flags.

Whether you are running puppet-access on a PE-managed server or installing it on a separate workstation, you need a global configuration file and a user-specified configuration file.

Global configuration file

The global configuration file is located at:

- **On *nix systems:** /etc/puppetlabs/client-tools/puppet-access.conf
- **On Windows systems:** `C:/ProgramData/PuppetLabs/client-tools/puppet-access.conf`

On machines managed by Puppet Enterprise, this global configuration file is created for you. The configuration file is formatted in JSON. For example:

```json
{
  "service-url": "https://<CONSOLE_HOSTNAME>:4433/rbac-api",
  "token-file": "~/.puppetlabs/token",
  "certificate-file": "/etc/puppetlabs/puppet/ssl/certs/ca.pem"
}
```

PE determines and adds the `service-url` setting.

If you're running `puppet-access` from a workstation not managed by PE, you must create the global file and populate it with the configuration file settings.

**User-specified configuration file**

The user-specified configuration file is located at `~/.puppetlabs/client-tools/puppet-access.conf` for both *nix and Windows systems. You must create the user-specified file and populate it with the configuration file settings. A list of configuration file settings is found in the next section.

The user-specified configuration file always takes precedence over the global configuration file. For example, if the two files have contradictory settings for the `token-file`, the user-specified settings prevail.

**Important:** User-specified configuration files must be in JSON format; HOCON and INI-style formatting are not supported.

**Configuration file settings for puppet-access**

As needed, you can manually add configuration settings to your user-specified or global `puppet-access` configuration file.

The class that manages the global configuration file is `puppet_enterprise::profile::controller`.

You can also change configuration settings by specifying flags when using the `puppet-access` command in the command line.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Command line flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>token-file</td>
<td>The location for storing authentication tokens. Defaults to <code>~/.puppetlabs/token</code>.</td>
<td><code>-t,--token-file</code></td>
</tr>
<tr>
<td>certificate-file</td>
<td>The location of the CA that signed the console-services server's certificate. Defaults to the PE CA cert location, <code>~/etc/puppetlabs/puppet/ssl/certs/ca.pem</code>.</td>
<td><code>--ca-cert</code></td>
</tr>
<tr>
<td>config-file</td>
<td>Changes the location of your configuration file. Defaults to <code>~/.puppetlabs/client-tools/puppet-access.conf</code>.</td>
<td><code>-c,--config-file</code></td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
<td>Command line flag</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>service-url</td>
<td>The URL for your RBAC API. Defaults to the URL automatically determined during the client tools package installation process, generally https://&lt;CONSOLE HOSTNAME&gt;:4433/rbac-api. You typically need to change this only if you are moving the console server.</td>
<td>--service-url</td>
</tr>
</tbody>
</table>

Generate a token using puppet-access

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Before you begin

Install the PE client tools package and configure `puppet-access`.

1. Choose one of the following options, depending on how long you need your token to last:
   - To generate a token with the default one-hour lifetime, run `sudo puppet-access login`.
   - To generate a token with a specific lifetime, run `sudo puppet-access login --lifetime <TIME Period>`. For example, `puppet-access login --lifetime 5h` generates a token that lasts five hours.

2. When prompted, enter the same username and password that you use to log into the PE console. The `puppet-access` command contacts the token endpoint in the RBAC v1 API. If your login credentials are correct, the RBAC service generates a token.

3. The token is generated and stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. You can print the token at any time using `puppet-access show`.

   You can continue to use this token until it expires, or until your access is revoked. The token has the exact same set of permissions as the user that generated it.

   **CAUTION:** If you run the login command with the `--debug` flag, the client outputs the token, as well as the username and password. For security reasons, exercise caution when using the `--debug` flag with the login command.

   **Note:** If a remote user generates a token and then is deleted from your external directory service, the deleted user cannot log into the console. However, because the token has already been authenticated, the RBAC service does not contact the external directory service again when the token is used in the future. To fully remove the token's access, you need to manually revoke or delete the user from PE.

   **Related information**
   - [Setting a token-specific lifetime](#) on page 351
     Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

   **Generate a token using the API endpoint**
   - The RBAC v1 API includes a token endpoint, which allows you to generate a token using curl commands.
1. On the command line, post your RBAC user login credentials using the token endpoint.

   ```
   type_header='Content-Type: application/json'
   uri="https://$(puppet config print server):4433/rbac-api/v1/auth/token"
   data='{"login": "<USER>",
   "password": "<PASSWORD>"}"
   
curl --insecure --header "$type_header" --request POST "$uri" --data "$data"
   
   See [Usage notes for curl examples](#) for information about forming curl commands.

   The command returns a JSON object containing the key `token` and the token itself.

   **CAUTION:** If you are using curl commands with the `--k` or `--insecure` SSL connection options, keep in mind that you are vulnerable to a person-in-the-middle attack.

2. Save the token. Depending on your workflow, either:

   - Copy the token to a text file.
   - Save the token as an environment variable using `export TOKEN=<PASTE THE TOKEN HERE>`.

   You can continue to use this token until it expires, or until your access is revoked. The token has the exact same set of permissions as the user that generated it.

   **Note:** If a remote user generates a token and then is deleted from your external directory service, the deleted user cannot log into the console. However, because the token has already been authenticated, the RBAC service does not contact the external directory service again when the token is used in the future. To fully remove the token's access, you need to manually revoke or delete the user from PE.

### Use a token with the PE API endpoints

The example below shows how to use a token in an API request. In this example, you use the `/users/current` endpoint of the RBAC v1 API to get information about the current authenticated user.

**Before you begin**

Generate a token using `puppet-access login`.

Run the following commands: `curl --insecure --header "X-Authentication: $(puppet-access show)" https://$(puppet config print server):4433/rbac-api/v1/users/current`

```auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current"
curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The commands above uses the X-Authentication header to supply the token information.

In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter by specifying the request as follows:

```uri="https://$(puppet config print server):4433/rbac-api/v1/users/current?token=$(puppet-access show)"
curl --insecure "$uri"
```

**CAUTION:** If you are using curl commands with the `--k` or `--insecure` SSL connection options, keep in mind that you are vulnerable to a person-in-the-middle attack.
Generate a token for use by a service

If you need to generate a token for use by a service and the token doesn't need to be saved, use the `--print` option.

**Before you begin**

Install the PE client tools package and configure `puppet-access`.

Run `sudo puppet-access login [username] --print`.

This command generates a token, and then displays the token content as stdout (standard output) rather than saving it to disk.

**Tip:** When generating a token for a service, consider specifying a longer token lifetime so that you don't have to regenerate the token too frequently.

Change the token's default lifetime

Tokens have a default authentication lifetime of one hour, but this default value can be adjusted in the console. You can also change a token's maximum authentication lifetime, which defaults to 10 years.

1. In the console, click Node groups.
2. Open the PE Infrastructure node group and click the PE Console node group.
3. On the Classes tab, find the `puppet_enterprise::profile::console` class.
4. In the Parameter field, select `rbac_token_auth_lifetime` to change the default lifetime of all tokens, or `rbac_token_maximum_lifetime` to adjust the maximum allowable lifetime for all tokens.
5. In the Value field, enter the new default authentication lifetime. Specify a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). For example, "12h" generates a token valid for 12 hours.

Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds.

The `rbac_token_auth_lifetime` cannot exceed the `rbac_token_maximum_lifetime` value.

6. Click Add parameter, and commit changes.

Setting a token-specific lifetime

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

Specify a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). For example, "12h" generates a token valid for 12 hours.

Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds.

Use the `--lifetime` parameter if using `puppet-access` to generate your token. For example: `puppet-access login --lifetime 1h`.

Use the `lifetime` value if using the RBAC v1 API to generate your token. For example: `{ "login": "<YOUR PE USER NAME>", "password": "<YOUR PE PASSWORD>", "lifetime": "1h" }`.

Change account expiration settings

In addition to token lifetime defaults, you can also activate and change the user account expiration and idle user settings.

**account-expiry-days**

This parameter is a positive integer that specifies the duration, in days, before an inactive user account expires. The default value is undefined.

© 2022 Puppet, Inc., a Perforce company
If a non-superuser hasn't logged into the console during this specified period, their user status updates to revoked. After creating an account, if a non-superuser hasn't logged in to the console during the specified period, their user status updates to revoked.

**Important:** If the `account-expiry-days` parameter is not specified, or has a value of less than 1, the `account-expiry-check-minutes` parameter is ignored.

To activate this setting in the console, add a value of 1 (day) or greater to the `account_expiry_days` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** group.

### account-expiry-check-minutes

This parameter is a positive integer that specifies how often, in minutes, the application checks for idle user accounts. The default value is 60 minutes.

To change this setting in the console, set a value (in minutes) for the `account_expiry_check_minutes` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** group.

For more information on configuration options, or to use a different configuration method for changing settings, see the [Methods for configuring PE](#) on page 286 docs.

---

### Set a token-specific label

You can affix a plain-text, user-specific label to tokens you generate with the RBAC v1 API. Token labels allow you to more readily refer to your token when working with RBAC API endpoints, or when revoking your own token.

Token labels are assigned on a per-user basis: two users can each have a token labelled “my token”, but a single user cannot have two tokens both labelled “my token.” You cannot use labels to refer to other users’ tokens.

Generate a token using the `token` endpoint of the RBAC API, using the `label` value to specify the name of your token.

```json
{
"login": "<YOUR PE USER NAME>",
"password": "<YOUR PE PASSWORD>",
"label": "Ava's token"
}
```

Labels must be no longer than 200 characters, must not contain commas, and must contain something besides whitespace.

---

### View token activity

Token activity is logged by the activity service. You can see recent token activity on any user's account in the console.

1. In the console, on the **Access control** page, click the **Users** tab and select the full name of the user you are interested in.
2. Click the **Activity** tab.

### Revoking a token

Revoke tokens by username, label, or full token through the `token` endpoint of the v2 RBAC API. All token revocation attempts are logged in the activity service, and can be viewed on the user's **Activity** tab in the console.

You can revoke your own token by username, label, or full token. You can also revoke any other full token you possess. Users with the permission to revoke other users can also revoke those users' tokens, as the `users:disable` permission includes token revocation. Revoking a user's tokens does not revoke the user's PE account.

If a user's account is revoked, all tokens associated with that user account are also automatically revoked.
Delete a token file

If you logged into puppet-access to generate a token, you can remove the file that stores that token simply by running the delete-token-file command. This is useful if you are working on a server that is used by other people.

Deleting the token file prevents other users from using your authentication token, but does not actually revoke the token. After the token has expired, there's no risk of obtaining the contents of the token file.

From the command line, run one of the following commands, depending on the path to your token file:
- If your token is at the default token file location, run puppet-access delete-token-file.
- If you used a different path to store your token file, run puppet-access delete-token-file --token-path <YOUR TOKEN PATH>.

RBAC API

Use the RBAC API v1 and v2 endpoints to manage users, roles, tokens, password, and LDAP connections.

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>Manage local users as well as those from a directory service, get lists of users, and create new local users.</td>
</tr>
<tr>
<td>groups</td>
<td>Get lists of groups and add a new remote user group.</td>
</tr>
<tr>
<td>roles</td>
<td>Get lists of user roles and create new roles.</td>
</tr>
<tr>
<td>permissions</td>
<td>Get information about available objects and the permissions that can be constructed for those objects.</td>
</tr>
<tr>
<td>ds (Directory service)</td>
<td>Get information about the directory service, test your directory service connection, and replace directory service connection settings.</td>
</tr>
<tr>
<td>password</td>
<td>Generate password reset tokens and update user passwords.</td>
</tr>
<tr>
<td>tokens</td>
<td>Generate the authentication tokens used to access PE. The v1/tokens endpoints are for creating tokens, while the v2/tokens endpoints are for revoking them.</td>
</tr>
<tr>
<td>rbac-service</td>
<td>Check the status of the RBAC service.</td>
</tr>
</tbody>
</table>

- **Forming RBAC API requests** on page 354
  
  Token-based authentication is required to access the RBAC API v1 and v2 endpoints. You can authenticate requests by using either user authentication tokens or allowed certificates.

- **RBAC service errors** on page 355
  
  You’re likely to encounter some errors when using the RBAC API. You’ll want to familiarize yourself with the error response descriptions and the general error responses.

- **RBAC API v1** on page 358
  
  The role-based access control (RBAC) API v1 service enables you to manage users, directory groups, and roles.

- **RBAC API v2** on page 378
  
  The role-based access control (RBAC) API v2 service enables you to validate groups and revoke tokens.

**Related information**

- **Status API** on page 436
The status API allows you to check the health of PE components and services. It can be useful for automated monitoring of your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

Configure RBAC and token-based authentication settings on page 295
Tune RBAC and token-based authentication settings, like setting the number of failed attempts a user has before they are locked out of the console or changing the amount of time a token is valid for.

Forming RBAC API requests
Token-based authentication is required to access the RBAC API v1 and v2 endpoints. You can authenticate requests by using either user authentication tokens or allowed certificates.

By default, the RBAC service listens on port 4433. All endpoints are relative to the /rbac-api/ path. So, for example, the full URL for the /v1/users endpoint on localhost is https://localhost:4433/rbac-api/v1/users.

Authenticate using tokens
Insert a user authentication token in an RBAC API request.
1. Generate a token: puppet-access login
2. Print the token and copy it: puppet-access show
3. Save the token as an environment variable: export TOKEN=<PASTE THE TOKEN HERE>
4. Include the token variable in your API request:

```bash
auth_header="X-Authentication: $(puppet-access show)"
uri=https://$(puppet config print server):4433/rbac-api/v1/users/current"
curl --insecure --header "$auth_header" "$uri"
```

See Usage notes for curl examples for information about forming curl commands.

The example above uses the X-Authentication header to supply the token information. In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter. To supply the token in a token parameter, specify the request as follows:

```bash
uri=https://$(puppet config print server):4433/rbac-api/v1/users/current?token=$(puppet-access show)"
curl --insecure "$uri"
```

⚠️ CAUTION: Be aware when using the token parameter method that the token parameter might be recorded in server access logs.

Authenticate using an allowed certificate
You can also authenticate requests using a certificate listed in RBAC's certificate allowlist, located at /etc/puppetlabs/console-services/rbac-certificate-allowlist. Note that if you edit this file, you must reload the pe-console-services service (run sudo service pe-console-services reload) for your changes to take effect.

Attach the certificate using the command line, as demonstrated in the example curl query below. You must have the allowed certificate name (which must match a name in the /etc/puppetlabs/console-services/rbac-certificate-allowlist file) and the private key to run the script.

```bash
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri=https://$(puppet config print server):4433/rbac-api/v1/users/current"
```
You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate specifically for use with the API.

**Related information**

Token-based authentication on page 347

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

**Content-type headers in the RBAC API**

RBAC accepts only JSON payloads in PUT and POST requests.

If a payload is provided, it is important to specify that the content is in JSON format. Thus, all PUT and POST requests with non-empty bodies should have the `Content-Type` header set to `application/json`.

**RBAC service errors**

You’re likely to encounter some errors when using the RBAC API. You’ll want to familiarize yourself with the error response descriptions and the general error responses.

**Error response format**

When the client specifies an `accept` header in the request with type `application/json`, the RBAC service returns errors in a standard format.

Each response is an object containing the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>A string classifying the error. It should be the same for all errors that have the same type of information in their <code>details</code> key.</td>
</tr>
<tr>
<td>msg</td>
<td>A human-readable message describing the error.</td>
</tr>
<tr>
<td>details</td>
<td>Additional machine-readable information about the error condition. The format of this key's value varies between kinds of errors, but is the same for each kind of error.</td>
</tr>
</tbody>
</table>

When returning errors in `text/html`, the body is the contents of the `msg` field.

**General error responses**

Any endpoint accepting a JSON body can return several kinds of 400 Bad Request responses.

<table>
<thead>
<tr>
<th>Response</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>malformed-request</td>
<td>400</td>
<td>The submitted data is not valid JSON. The <code>details</code> key consists of one field, <code>error</code>, which contains the error message from the JSON parser.</td>
</tr>
<tr>
<td>Response</td>
<td>Status</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>schema-violation</td>
<td>400</td>
<td>The submitted data has an unexpected structure, such as invalid fields or missing required fields. The <code>msg</code> contains a description of the problem. The <code>details</code> are an object with three keys:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>submitted</code>: The submitted data as it was seen during schema validation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>schema</code>: The expected structure of the data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>error</code>: A structured description of the error.</td>
</tr>
<tr>
<td>inconsistent-id</td>
<td>400</td>
<td>Data was submitted to an endpoint where the ID of the object is a part of the URL and the submitted data contains an <code>id</code> field with a different value. The <code>details</code> key consists of two fields, <code>url-id</code> and <code>body-id</code>, showing the IDs from both sources.</td>
</tr>
<tr>
<td>invalid-id-filter</td>
<td>400</td>
<td>A URL contains a filter on the ID with an invalid format. No details are given with this error.</td>
</tr>
<tr>
<td>invalid-uuid</td>
<td>400</td>
<td>An invalid UUID was submitted. No details are given with this error.</td>
</tr>
<tr>
<td>user-unauthenticated</td>
<td>401</td>
<td>An unauthenticated user attempted to access a route that requires authentication.</td>
</tr>
<tr>
<td>user-revoked</td>
<td>401</td>
<td>A user who has been revoked attempted to access a route that requires authentication.</td>
</tr>
<tr>
<td>api-user-login</td>
<td>401</td>
<td>A person attempted to log in as the <code>api_user</code> with a password (<code>api_user</code> does not support username/password authentication).</td>
</tr>
<tr>
<td>Response</td>
<td>Status</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remote-user-conflict</td>
<td>401</td>
<td>A remote user who is not yet known to RBAC attempted to authenticate, but a local user with that login already exists.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The solution is to change either the local user's login in RBAC, or to change the remote user's login, either by changing the user_lookup_attr in the DS settings or by changing the value in the directory service itself.</td>
</tr>
<tr>
<td>permission-denied</td>
<td>403</td>
<td>A user attempted an action that they are not permitted to perform.</td>
</tr>
<tr>
<td>admin-user-immutable</td>
<td>403</td>
<td>A user attempted to edit metadata or associations belonging to the default roles (&quot;Administrators&quot;, &quot;Operators&quot;, &quot;Code Deployers&quot;, or &quot;Viewers&quot;) or default users (&quot;admin&quot; or &quot;api_user&quot;) that they are not allowed to change.</td>
</tr>
<tr>
<td>admin-user-not-in-admin-role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default-roles-immutable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conflict</td>
<td>409</td>
<td>A value for a field that is supposed to be unique was submitted to the service and another object has that value. For example, when a user is created with the same login as an existing user.</td>
</tr>
<tr>
<td>invalid-associated-id</td>
<td>422</td>
<td>An object was submitted with a list of associated IDs (for example, user_ids) and one or more of those IDs does not correspond to an object of the correct type.</td>
</tr>
<tr>
<td>no-such-user-LDAP</td>
<td>422</td>
<td>An object was submitted with a list of associated IDs (for example, user_ids) and one or more of those IDs does not correspond to an object of the correct type.</td>
</tr>
<tr>
<td>no-such-group-LDAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-unique-lookup-attr</td>
<td>422</td>
<td>A login was attempted but multiple users are found via LDAP for the given username. The directory service settings must use a user_lookup_attr that is guaranteed to be unique within the provided user's RDN.</td>
</tr>
<tr>
<td>server-error</td>
<td>500</td>
<td>Occurs when the server throws an unspecified exception. A message and stack trace should be available in the logs.</td>
</tr>
</tbody>
</table>
**RBAC API v1**

The role-based access control (RBAC) API v1 service enables you to manage users, directory groups, and roles.

- **Users endpoints** on page 358
  RBAC enables you to manage local users as well as those who are created remotely, on a directory service. With the users endpoints, you can get lists of users and create new local users.

- **User group endpoints** on page 363
  Groups are used to assign roles to a group of users, which is vastly more efficient than managing roles for each user individually. The groups endpoints enable you to get lists of groups, and to add a new directory group.

- **User roles endpoints** on page 366
  By assigning roles to users, you can manage them in sets that are granted access permissions to various PE objects. This makes tracking user access more organized and easier to manage. The roles endpoints enable you to get lists of roles and create new roles.

- **Permissions endpoints** on page 369
  You assign permissions to user roles to manage user access to objects. The permissions endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

- **Token endpoints** on page 372
  A user's access to PE services can be controlled using authentication tokens. Users can generate their own authentication tokens using the token endpoint.

- **Directory service endpoints** on page 374
  Use the ds (directory service) API endpoints to get information about the directory service, test your directory service connection, and replace directory service connection settings.

- **Password endpoints** on page 376
  When local users forget passwords or lock themselves out of PE by attempting to log in with incorrect credentials too many times, you must generate a password reset token for them. The password endpoints enable you to generate password reset tokens for a specific local user or with a token that contains a temporary password in the body.

**Related information**

API index on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Users endpoints**

RBAC enables you to manage local users as well as those who are created remotely, on a directory service. With the users endpoints, you can get lists of users and create new local users.

- **Users keys** on page 359
  The following keys are used with the RBAC v1 API's users endpoints.

- **GET /users** on page 359
  Fetches all users, both local and remote (including the superuser). Supports filtering by ID through query parameters. Authentication is required.

- **GET /users/<sid>** on page 360
  Fetches a single user by its subject ID (sid). Authentication is required.

- **GET /users/current** on page 361
  Fetches the data about the current authenticated user, with the exact same behavior as GET /users/<sid>, except that <sid> is assumed from the authentication context. Authentication is required.

- **POST /users** on page 361
  Creates a new local user. You can add the new user to user roles by specifying an array of roles in role_ids. You can set a password for the user in password. For the password to work in the PE console, it needs to be a minimum of six characters. Authentication is required.

- **PUT /users/<sid>** on page 361
  Replaces the user with the specified ID (sid) with a new user object. Authentication is required.

- **DELETE /users/<sid>** on page 362
  Deletes the user with the specified ID (sid), regardless of whether they are a user defined in RBAC or a user defined by a directory service. In the case of directory service users, while this action removes a user from the console, that
user is still able to log in (at which point they are re-added to the console) if they are not revoked. Authentication is required.

**Users keys**
The following keys are used with the RBAC v1 API's `users` endpoints.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>A UUID string identifying the user.</td>
<td>&quot;4fee7450-54c7-11e4-916c-0800200c9a66&quot;</td>
</tr>
<tr>
<td>login</td>
<td>A string used by the user to log in.</td>
<td>&quot;admin&quot;</td>
</tr>
<tr>
<td>email</td>
<td>An email address string.</td>
<td>&quot;<a href="mailto:hill@example.com">hill@example.com</a>&quot;</td>
</tr>
<tr>
<td>display_name</td>
<td>The user's name as a string.</td>
<td>&quot;Kalo Hill&quot;</td>
</tr>
<tr>
<td>role_ids</td>
<td>An array of role IDs indicating which roles should be directly assigned to the user. An empty array is valid.</td>
<td>[3 6 5]</td>
</tr>
<tr>
<td>is_group</td>
<td>These flags indicate the type of user.</td>
<td>true/false</td>
</tr>
<tr>
<td>is_remote</td>
<td>is_group should always be false for a user.</td>
<td></td>
</tr>
<tr>
<td>is_superuser</td>
<td>Setting this flag to true prevents the user from accessing any routes until the flag is unset or the user's password is reset via token.</td>
<td>true/false</td>
</tr>
<tr>
<td>is_revoked</td>
<td>This is a timestamp in UTC-based ISO-8601 format (YYYY-MM-DDThh:mm:ssZ) indicating when the user last logged in. If the user has never logged in, this value is null.</td>
<td>&quot;2014-05-04T02:32:00Z&quot;</td>
</tr>
<tr>
<td>group_ids</td>
<td>An array of UUIDs indicating which groups a remote user inherits roles from.</td>
<td>[&quot;3a96d280-54c9-11e4-916c-0800200c9a66&quot;]</td>
</tr>
<tr>
<td>inherited_role_ids</td>
<td>An array of role IDs indicating which roles a remote user inherits from their groups.</td>
<td>[9 1 3]</td>
</tr>
</tbody>
</table>

**GET /users**
Fetched all users, both local and remote (including the superuser). Supports filtering by ID through query parameters. Authentication is required.

**Request format**
To request all the users:

```
GET /users
```

To request specific users, add a comma-separated list of user IDs:

```
GET /users?
id=fe62d770-5886-11e4-8ed6-0800200c9a66,1cadd0e0-5887-11e4-8ed6-0800200c9a66
```
Response format

The response is a JSON object that lists the metadata for all requested users.

For example:

```json
[
  {
    "id": "fe62d770-5886-11e4-8ed6-0800200c9a66",
    "login": "Kalo",
    "email": "kalohill@example.com",
    "display_name": "Kalo Hill",
    "role_ids": [1,2,3...],
    "is_group" : false,
    "is_remote" : false,
    "is_superuser" : true,
    "is_revoked": false,
    "last_login": "2014-05-04T02:32:00Z"
  },
  {
    "id": "07d9c8e0-5887-11e4-8ed6-0800200c9a66",
    "login": "Jean",
    "email": "jeanjackson@example.com",
    "display_name": "Jean Jackson",
    "role_ids": [2,3],
    "inherited_role_ids": [5],
    "is_group" : false,
    "is_remote" : true,
    "is_superuser" : false,
    "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
    "is_revoked": false,
    "last_login": "2014-05-04T02:32:00Z"
  },
  {
    "id": "1cadd0e0-5887-11e4-8ed6-0800200c9a66",
    "login": "Amari",
    "email": "amariperez@example.com",
    "display_name": "Amari Perez",
    "role_ids": [2,3],
    "inherited_role_ids": [5],
    "is_group" : false,
    "is_remote" : true,
    "is_superuser" : false,
    "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
    "is_revoked": false,
    "last_login": "2014-05-04T02:32:00Z"
  }
]
```

GET /users/<sid>

Fetches a single user by its subject ID (sid). Authentication is required.

Response format

For all users, the user contains an ID, a login, an email, a display name, a list of role-ids the user is directly assigned to, and the last login time in UTC-based ISO-8601 format (YYYY-MM-DDThh:mm:ssZ), or null if the user has not logged in yet. It also contains an "is_revoked" field, which, when set to true, prevents a user from authenticating.

For example:

```json
{"id": "fe62d770-5886-11e4-8ed6-0800200c9a66",
 "login": "Amari",
 "email": "amariperez@example.com",
 "display_name": "Amari Perez",
 "role_ids": [1,2,3...],
 "is_group" : false,
 "is_remote" : false,
}
"is_superuser" : false,
"is_revoked": false,
"last_login": "2014-05-04T02:32:002"

For remote users, the response additionally contains a field indicating the IDs of the groups the user inherits roles from and the list of inherited role IDs.

For example:

```
{"id": "07d9c8e0-5887-11e4-8ed6-0800200c9a66",
"login": "Jean",
"email": "jeanjackson@example.com",
"display_name": "Jean Jackson",
"role_ids": [2,3...],
"inherited_role_ids": [],
"is_group" : false,
"is_remote" : true,
"is_superuser" : false,
"group_ids": ["b28b8790-5889-11e4-8ed6-0800200c9a66"],
"is_revoked": false,
"last_login": "2014-05-04T02:32:002"}
```

GET /users/current

Fetches the data about the current authenticated user, with the exact same behavior as GET /users/<sid>, except that <sid> is assumed from the authentication context. Authentication is required.

POST /users

Creates a new local user. You can add the new user to user roles by specifying an array of roles in role_ids. You can set a password for the user in password. For the password to work in the PE console, it needs to be a minimum of six characters. Authentication is required.

Request format

Accepts a JSON body containing entries for email, display_name, login, role_ids, and password. The password field is optional. The created account is not useful until the password is set.

For example:

```
{"login":"Kalo",
"email":"kalohill@example.com",
"display_name":"Kalo Hill",
"role_ids": [1,2,3],
"password": "yabbadabba"}
```

Response format

If the create operation is successful, a 201 Created response with a location header that points to the new resource is returned.

Error responses

If the email or login for the user conflicts with an existing user's login, a 409 Conflict response is returned.

PUT /users/<sid>

Replaces the user with the specified ID (sid) with a new user object. Authentication is required.

Request format

Accepts an updated user object with all keys provided when the object is received from the API. The behavior varies based on user type. All types have a role_ids array that indicates all the roles the user should belong to. An empty roles array removes all roles directly assigned to the group.
The below examples show what keys must be submitted. Keys marked with asterisks are the only ones that can be changed via the API.

An example for a local user:

```
{"id": "c8b2c380-5889-11e4-8ed6-0800200c9a66",
"login": "Amari",
"email": "amari.perez@example.com",
"display_name": "Amari Perez",
"role_ids": [1, 2, 3],
"is_group": false,
"is_remote": false,
"is_superuser": false,
"is_revoked": false,
"last_login": "2014-05-04T02:32:00Z"}
```

An example for a remote user:

```
{"id": "3271fde0-588a-11e4-8ed6-0800200c9a66",
"login": "Jean",
"email": "jean.jackson@example.com",
"display_name": "Jean Jackson",
"role_ids": [4, 1],
"inherited_role_ids": [],
"group_ids": [],
"is_group": false,
"is_remote": true,
"is_superuser": false,
"is_revoked": false,
"last_login": "2014-05-04T02:32:00Z"}
```

**Response format**

The request returns a 200 OK response, along with the user object with the changes made.

**Error responses**

If the login for the user conflicts with an existing user login, a 409 Conflict response is returned.

**DELETE /users/<sid>**

Deletes the user with the specified ID (sid), regardless of whether they are a user defined in RBAC or a user defined by a directory service. In the case of directory service users, while this action removes a user from the console, that user is still able to log in (at which point they are re-added to the console) if they are not revoked. Authentication is required.

**Remember:** The admin user and the api_user cannot be deleted.

**Request format**

For example, to delete a user with the ID 3982a629-1278-4e38-883e-12a7cac91535 by using a curl command:

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/3982a629-1278-4e38-883e-12a7cac91535"
curl --cert "$cert" --cacert "$cacert" --key "$key" --request DELETE "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.
Response format

If the user is successfully deleted, a 204 No Content response with an empty body is returned.

Error responses

If the current user does not have the `users:edit` permission for this user group, a 403 Forbidden response is returned.

If a user with the provided identifier does not exist, a 404 Not Found response is returned.

User group endpoints

Groups are used to assign roles to a group of users, which is vastly more efficient than managing roles for each user individually. The groups endpoints enable you to get lists of groups, and to add a new directory group.

Remember: Group membership is determined by the directory service hierarchy and as such, local users cannot be in directory groups.

- **User group keys** on page 363
  The following keys are used with the RBAC v1 API's groups endpoints.
  - **GET /groups** on page 364
    Fetches all groups. Supports filtering by ID through query parameters. Authentication is required.
  - **GET /groups/<sid>** on page 365
    Fetches a single group by its subject ID (sid). Authentication is required.
  - **POST /groups** on page 365
    Creates a new remote group and attaches to the new group any roles specified in its roles list. Authentication is required.
  - **PUT /groups/<sid>** on page 366
    Replaces the group with the specified ID (sid) with a new group object. Authentication is required.
  - **DELETE /groups/<sid>** on page 366
    Deletes the user group with the specified ID (sid) from RBAC without making any changes to the directory service. Authentication required.

User group keys

The following keys are used with the RBAC v1 API's groups endpoints.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>A UUID string identifying the group.</td>
<td>&quot;c099d420-5557-11e4-916c-0800200c9a66&quot;</td>
</tr>
<tr>
<td>login</td>
<td>the identifier for the user group on the directory server.</td>
<td>&quot;poets&quot;</td>
</tr>
<tr>
<td>display_name</td>
<td>The group's name as a string.</td>
<td>&quot;Poets&quot;</td>
</tr>
<tr>
<td>role_ids</td>
<td>An array of role IDs indicating which roles should be inherited by the group's members. An empty array is valid. This is the only field that can be updated via RBAC; the rest are immutable or synced from the directory service.</td>
<td>[3 6 5]</td>
</tr>
<tr>
<td>is_group</td>
<td>These flags indicate that the group is a group.</td>
<td>true, true, false, respectively</td>
</tr>
<tr>
<td>is_remote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_superuser</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Key Explanation

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_revoked</td>
<td>Setting this flag to <code>true</code> currently does nothing for a group.</td>
<td><code>true/false</code></td>
</tr>
<tr>
<td>user_ids</td>
<td>An array of UUIDs indicating which users inherit roles from this group.</td>
<td><code>&quot;3a96d280-54c9-11e4-916c-0800200c9a66&quot;</code></td>
</tr>
</tbody>
</table>

### GET /groups

Fetches all groups. Supports filtering by ID through query parameters. Authentication is required.

#### Request format

The following requests all the groups:

```
GET /groups
```

To request only some groups, add a comma-separated list of group IDs:

```
GET /groups?
id=65a068a0-588a-11e4-8ed6-0800200c9a66,75370a30-588a-11e4-8ed6-0800200c9a66
```

#### Response format

The response is a JSON object that lists the metadata for all requested groups.

For example:

```json
[
{
  "id": "65a068a0-588a-11e4-8ed6-0800200c9a66",
  "login": "hamsters",
  "display_name": "Hamster club",
  "role_ids": [2, 3],
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids": ["07d9c8e0-5887-11e4-8ed6-0800200c9a66"]
},
{
  "id": "75370a30-588a-11e4-8ed6-0800200c9a66",
  "login": "chinchilla",
  "display_name": "Chinchilla club",
  "role_ids": [2, 1],
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids": ["07d9c8e0-5887-11e4-8ed6-0800200c9a66"]
 },
{
  "id": "ccdbde50-588a-11e4-8ed6-0800200c9a66",
  "login": "wombats",
  "display_name": "Wombat club",
  "role_ids": [2, 3],
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids": []
}
]```
GET /groups/<sid>
Fetches a single group by its subject ID (sid). Authentication is required.

**Response format**
The response contains an ID for the group and a list of *role_ids* the group is directly assigned to.

For directory groups, the response contains the display name, the login field, a list of *role_ids* directly assigned to the group, and *user_ids* containing IDs of the remote users that belong to that group.

For example:

```json
{"id": "65a068a0-588a-11e4-8ed6-0800200c9a66",
 "login": "hamsters",
 "display_name": "Hamster club",
 "role_ids": [2, 3],
 "is_group": true,
 "is_remote": true,
 "is_superuser": false,
 "user_ids": ["07d9c8e0-5887-11e4-8ed6-0800200c9a66"]}
```

**Error responses**
If the user who submits the GET request has not successfully authenticated, a 401 Unauthorized response is returned.

If the current user does not have the appropriate user permissions to request the group data, a 403 Forbidden response is returned.

**POST /groups**
Creates a new remote group and attaches to the new group any roles specified in its roles list. Authentication is required.

**Request format**
Accepts a JSON body containing an entry for *login*, and an array of *role_ids* to assign to the group initially.

For example:

```json
{"login": "Augmentators",
 "role_ids": [1, 2, 3]}
```

**Response format**
If the create operation is successful, a 201 Created response with a location header that points to the new resource is returned.

**Error responses**
If the login for the group conflicts with an existing group login, a 409 Conflict response is returned.

**PUT /groups/<sid>**
Replaces the group with the specified ID (sid) with a new group object. Authentication is required.

**Request format**
Accepts an updated group object containing all the keys that were received from the API initially. The only updatable field is *role_ids*. An empty roles array indicates a desire to remove the group from all the roles it was directly assigned to. Any other changed values are ignored.
For example:

```json
{
    "id": "75370a30-588a-11e4-8ed6-0800200c9a66",
    "login": "chinchilla",
    "display_name": "Chinchillas",
    "role_ids": [2, 1],
    "is_group": true,
    "is_remote": true,
    "is_superuser": false,
    "user_ids": [
        "1cadd0e0-5887-11e4-8ed6-0800200c9a66",
        "5c1ab4b0-588b-11e4-8ed6-0800200c9a66"
    ]
}
```

**Response format**

If the operation is successful, a 200 OK response including a group object with updated roles is returned.

**DELETE /groups/<sid>**

Deletes the user group with the specified ID (sid) from RBAC without making any changes to the directory service. Authentication required.

**Response format**

If the user group is successfully deleted, a 204 No Content with an empty body is returned.

**Error responses**

If the current user does not have the `user_groups:delete` permission for this user group, a 403 Forbidden response is returned.

If a group with the provided identifier does not exist, a 404 Not Found response is returned.

**User roles endpoints**

By assigning roles to users, you can manage them in sets that are granted access permissions to various PE objects. This makes tracking user access more organized and easier to manage. The roles endpoints enable you to get lists of roles and create new roles.

- **User role keys** on page 366

The following keys are used with the RBAC v1 API’s roles endpoints.

- **GET /roles** on page 367

Fetches all roles with user and group ID lists and permission lists. Authentication is required.

- **GET /roles/<rid>** on page 367

Fetches a single role by its ID (rid). Authentication is required.

- **POST /roles** on page 368

Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

- **PUT /roles/<rid>** on page 368

Replaces a role at the specified ID (rid) with a new role object. Authentication is required.

- **DELETE /roles/<rid>** on page 369

Deletes the role identified by the role ID (rid). Users with this role immediately lose the role and all permissions granted by it, but their session is otherwise unaffected. Access to the next request that the user makes is determined by the new set of permissions the user has without this role.

**User role keys**

The following keys are used with the RBAC v1 API’s roles endpoints.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An integer identifying the role.</td>
<td>18</td>
</tr>
<tr>
<td>display_name</td>
<td>The role’s name as a string.</td>
<td>&quot;Viewers&quot;</td>
</tr>
<tr>
<td>Key</td>
<td>Explanation</td>
<td>Example</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A string describing the role's function.</td>
<td>&quot;View-only permissions&quot;</td>
</tr>
<tr>
<td>permissions</td>
<td>An array containing permission objects that indicate what permissions a role grants. An empty array is valid. See Permission keys for more information.</td>
<td>[]</td>
</tr>
<tr>
<td>user_ids</td>
<td>An array of UUIDs indicating which users and groups are directly assigned to the role. An empty array is valid.</td>
<td>[&quot;fc115750-555a-11e4-916c-0800200c9a66&quot;]</td>
</tr>
<tr>
<td>group_ids</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GET /roles**

Fetches all roles with user and group ID lists and permission lists. Authentication is required.

**Response format**

Returns a JSON object containing all roles with user and group ID lists and permission lists.

For example:

```json
[{"id": 123,  "permissions": [{"object_type":"node_groups",  "action":"edit_rules",  "instance":"*"}],  "user_ids": ["1cadd0e0-5887-11e4-8ed6-0800200c9a66","5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],  "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],  "display_name": "A role",  "description": "Edit node group rules"}, ]
```

**GET /roles/<rid>**

Fetches a single role by its ID (rid). Authentication is required.

**Response format**

Returns a 200 OK response with the role object with a full list of permissions and user and group IDs.

For example:

```json
{"id": 123,  "permissions": [{"object_type":"node_groups",  "action":"edit_rules",  "instance":"*"}],  "user_ids": ["1cadd0e0-5887-11e4-8ed6-0800200c9a66","5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],  "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],  "display_name": "A role",  "description": "Edit node group rules"}
POST /roles
Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

Permissions keys for task-targets
If you're writing a role for a task-target, you must include unique action and instance key values to specify permissions. For the complete task-target workflow, see the blog post Puppet Enterprise RBAC API, or how to manage access to tasks.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>run_with_constraints</td>
<td>Specifies that the user has permission to run a task on certain nodes within the confines of a given task-target.</td>
</tr>
<tr>
<td>instance</td>
<td>&lt;task-target ID&gt;</td>
<td>Specifies the ID of the task-target the user has permission to run.</td>
</tr>
</tbody>
</table>

Request format
Accepts a new role object. Any of the arrays can be empty and "description" can be null.

For example:

```
{"permissions": [{"object_type":"node_groups", "action":"edit_rules", "instance":"*"}, ...],
"user_ids": ["1cadd0e0-5887-11e4-8ed6-0800200c9a66","5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],
"group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
"display_name": "A role",
"description": "Edit node group rules"}
```

Response format
Returns a 201 Created response with a location header pointing to the new resource.

Error responses
Returns a 409 Conflict response if the role has a name that collides with an existing role.

Related information
POST /command/task_target on page 708
Create a new task-target, which is a collection of tasks, nodes and node groups that define a permission group. When a user has permissions to run via a given task-target, they are granted permissions to run the given set of tasks, on the set of nodes described by the task-target.

PUT /roles/<rid>
Replaces a role at the specified ID (rid) with a new role object. Authentication is required.

Request format
Accepts the modified role object.

For example:

```
{"id": 123,
 "permissions": [{"object_type":"node_groups", "action":"edit_rules", "instance":"*"}, ...],
```
"user_ids": ["1cadd0e0-5887-11e4-8ed6-0800200c9a66","5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],
"group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
"display_name": "A role",
"description": "Edit node group rules"

Response format
Returns a 200 OK response with the modified role object.

Error responses
Returns a 409 Conflict response if the new role has a name that collides with an existing role.

DELETE /roles/<rid>
Deletes the role identified by the role ID (rid). Users with this role immediately lose the role and all permissions granted by it, but their session is otherwise unaffected. Access to the next request that the user makes is determined by the new set of permissions the user has without this role.

Response format
Returns a 200 OK response if the role identified by <rid> has been deleted.

Error responses
Returns a 404 Not Found response if no role exists for <rid>.
Returns a 403 Forbidden response if the current user lacks permission to delete the role identified by <rid>.

Permissions endpoints
You assign permissions to user roles to manage user access to objects. The permissions endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

• Permissions keys on page 369
The following keys are used with the RBAC v1 API’s permissions endpoints. The available values for these keys are available from the /types endpoint (see below).

• GET /types on page 370
Lists the objects that integrate with RBAC and demonstrates the permissions that can be constructed by picking the appropriate object_type, action, and instance triple. Authentication is required.

• POST /permitted on page 370
Checks an array of permissions for the subject identified by the submitted identifier.

• GET /permitted/<object-type>/<action> on page 371
Return an array of permitted instances for the given object-type/action pair based on the current user authentication context.

• GET /permitted/<object-type>/<action>/<uuid> on page 371
Return an array of permitted instances for the given object-type/action pair based on the user matching the supplied uuid.

Permissions keys
The following keys are used with the RBAC v1 API’s permissions endpoints. The available values for these keys are available from the /types endpoint (see below).

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>A string identifying the type of object this permission applies to.</td>
<td>&quot;node_groups&quot;</td>
</tr>
<tr>
<td>action</td>
<td>A string indicating the type of action this permission permits.</td>
<td>&quot;modify_children&quot;</td>
</tr>
</tbody>
</table>
### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>A string containing the primary ID of the object instance this permission applies to, or &quot;<em>&quot; indicating that it applies to all instances. If the given action does not allow instance specification, &quot;</em>&quot; should always be used.</td>
<td>&quot;cec7e830-555b-11e4-916c-0800200c9a66&quot; or &quot;*&quot;</td>
</tr>
</tbody>
</table>

### GET /types

Lists the objects that integrate with RBAC and demonstrates the permissions that can be constructed by picking the appropriate object_type, action, and instance triple. Authentication is required.

The `has_instances` flag indicates that the action permission is instance-specific if `true`, or `false` if this action permission does not require instance specification.

**Response format**

Returns a 200 OK response with a listing of types.

For example:

```json
[
  { "object_type": "node_groups",  
    "display_name": "Node Groups",  
    "description": "Groups that nodes can be assigned to."  
    "actions": [{  
      "name": "view",  
      "display_name": "View",  
      "description": "View the node groups",  
      "has_instances": true  
    },  
    "name": "modify",  
    "display_name": "Configure",  
    "description": "Modify description, variables and classes",  
    "has_instances": true  
    }, ...
  },...
]
```

**Error responses**

Returns a 401 Unauthorized response if no user is logged in.

Returns a 403 Forbidden response if the current user lacks permissions to view the types.

### POST /permitted

Checks an array of permissions for the subject identified by the submitted identifier.

**Request format**

This endpoint takes a "token" in the form of a user or a user group's UUID and a list of permissions. This returns true or false for each permission queried, representing whether the subject is permitted to take the given action.

The full evaluation of permissions is taken into account, including inherited roles and matching general permissions against more specific queries. For example, a query for `users:edit:1` returns `true` if the subject has `users:edit:1` or `users:edit:*`.

In the following example, the first permission is querying whether the subject specified by the token is permitted to perform the `edit_rules` action on the instance of `node_groups` identified by the ID 4. Note that in reality, node groups and users use UUIDs as their IDs.

```json
{"token": "<subject uuid>",
```
"permissions": [{"object_type": "node_groups", "action": "edit_rules", "instance": "4"}, {"object_type": "users", "action": "disable", "instance": "1"}]

Response format

Returns a 200 OK response with an array of Boolean values representing whether each submitted action on a specific object type and instance is permitted for the subject. The array always has the same length as the submitted array and each returned Boolean value corresponds to the submitted permission query at the same index.

The example response below was returned from the example request in the previous section. This return means the subject is permitted node_groups:edit_rules:4 but not permitted users:disable:1.

[true, false]

GET /permitted/<object-type>/<action>

Return an array of permitted instances for the given object-type/action pair based on the current user authentication context.

This endpoint returns the instance values as an array for the given object-type and action pair. If the user does not have permissions for any instance, an empty array will be returned.

If the object-type does not map to a known object-type, the endpoint will return a 404.

If the action does not exist for the given object-type, the endpoint will return a 404.

Example

GET /permitted/node_groups/view

["00000000-0000-4000-8000-000000000000"]

Returns a 200 OK response with a listing of instances.

GET /permitted/<object-type>/<action>/<uuid>

Return an array of permitted instances for the given object-type/action pair based on the user matching the supplied uuid.

This endpoint returns the instance values as an array for the given object-type and action pair. If the user does not have permissions for any instances, and empty array will be returned.

If the object-type does not map to a know object-type, the endpoint will return a 404.

If the action does not exist for the given object-type, the endpoint will return a 404.

If the uuid does not map to a known user, the endpoint will return a 404.

Example

GET /permitted/node_groups/view/fe62d770-5886-11e4-8ed6-0800200c9a66

["00000000-0000-4000-8000-000000000000"]

Returns a 200 OK response with a listing of instances.
**Token endpoints**
A user's access to PE services can be controlled using authentication tokens. Users can generate their own authentication tokens using the token endpoint.

- **Token keys** on page 372
  The following keys are used with the token endpoint.
- **POST /auth/token** on page 373
  Generates an access token for the user whose login information is POSTed. This token can then be used to authenticate requests to PE services using either the X-Authentication header or the token query parameter.

**Related information**
*Token-based authentication* on page 347
Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

**Token keys**
The following keys are used with the token endpoint.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>The user's login for the PE console (required).</td>
</tr>
<tr>
<td>password</td>
<td>The user's password for the PE console (required).</td>
</tr>
<tr>
<td>lifetime</td>
<td>The length of time the token is active before expiration (optional).</td>
</tr>
<tr>
<td>description</td>
<td>Additional metadata about the requested token (optional).</td>
</tr>
<tr>
<td>client</td>
<td>Additional metadata about the client making the token request (optional).</td>
</tr>
<tr>
<td>label</td>
<td>A user-defined label for the token (optional).</td>
</tr>
</tbody>
</table>

**The lifetime key**
When setting a token's lifetime, specify a numeric value followed by y (years), d (days), h (hours), m (minutes), or s (seconds). For example, a value of 12h is 12 hours. Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds. If you do not want the token to expire, set the lifetime to 0. Setting it to zero gives the token a lifetime of approximately 10 years.

**Tip:** The default lifetime for all tokens is also configurable. See Change the token's default lifetime for configuration instructions.

**The label key**
You can choose to select a label for the token that can be used with other RBAC token endpoints. Labels:

- Must be no longer than 200 characters.
- Must not contain commas.
- Must contain something other than whitespace. (Whitespace is trimmed from the beginning and end of the label, though it is allowed elsewhere.)
- Must not be the same as a label for another token for the same user.

Token labels are assigned on a per-user basis: two users can each have a token labelled *my token*, but a single user cannot have two tokens both labelled *my token*. You cannot use labels to refer to other users' tokens.
POST /auth/token
Generates an access token for the user whose login information is POSTed. This token can then be used to authenticate requests to PE services using either the X-Authentication header or the token query parameter.

This route is intended to require zero authentication. While HTTPS is still required (unless PE is explicitly configured to permit HTTP), neither an allowed cert nor a session cookie is needed to POST to this endpoint.

Request format
Accepts a JSON object or curl command with the user's login and password information. The token's lifetime, a user-specified label, and additional metadata can be added, but are not required.

An example JSON request:

```json
{
  "login": "jeanjackson@example.com",
  "password": "1234",
  "lifetime": "4m",
  "label": "personal workstation token"
}
```

An example curl command request:

```
type_header='Content-Type: application/json'
cacert="$(puppet config print cacert)"
uri="https://$(puppet config print server):4433/rbac-api/v1/auth/token"
data='{"login": "<USER>",
          "password": "<PASSWORD>",
          "lifetime": "4h",
          "label": "four-hour token"}'
curl --header "$type_header" --cacert "$cacert" --request POST "$uri" --data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

The various parts of this curl command request are explained as follows:

- **--header 'Content-Type: application/json'**: sets the Content-Type header to application/json, which indicates to RBAC that the data being sent is in JSON format.
- **--cacert [FILE]**: Specifies a CA certificate as described in Forming requests for the node classifier. Alternatively, you could use the --insecure flag to turn off SSL verification of the RBAC server so that you can use the HTTPS protocol without providing a CA cert. If you do not provide one of these options in your curl request, curl complains about not being able to verify the RBAC server.

**Note:** The --insecure flag is shown as an example only. You should use your own discretion when choosing the appropriate server verification method for the tool that you are using.

- **--request POST**: This is an HTTP POST request to provide your login information to the RBAC service.
- **https://<HOSTNAME>:<PORT>/rbac-api/v1/auth/token**: Sends the request to the token endpoint. For HOSTNAME, provide the FQDN of the server that is hosting the PE console service. If you are making the call from the console server, you can use "localhost." For PORT, provide the port that the PE services (node classifier service, RBAC service, and activity service) listen on. The default port is 4433
- **--data '{"login": "<USER>", "password": "<PASSWORD>", "lifetime": "4h",
    "label": "four-hour token"}'**: Provide the user name and password that you use to log in to the PE console. Optionally, set the token's lifetime and label.

Response format
Returns a 200 OK response if the credentials are good and the user is not revoked, along with a token.
For example:

```
{"token": "asd0u0=2jdijasodj-w0duwghjashd,kjsahdaso10d9hw0hduashd0a9wdy0whdkaudhaksdhc9chakdh92..."}
```

Error responses

Returns a 401 Unauthenticated response if the credentials are bad or the user is revoked.

Returns a 400 Malformed response if something is wrong with the request body.

Directory service endpoints

Use the `ds` (directory service) API endpoints to get information about the directory service, test your directory service connection, and replace directory service connection settings.

To connect to the directory service anonymously, specify `null` for the lookup user and lookup password or leave these fields blank.

- **GET /ds** on page 374
  Get the connected directory service information. Authentication is required.
- **GET /ds/test** on page 374
  Runs a connection test for the connected directory service. Authentication is required.
- **PUT /ds/test** on page 375
  Performs a connection test with the submitted settings. Authentication is required.
- **PUT /ds** on page 375
  Replaces current directory service connection settings. Authentication is required.

Related information

External directory settings on page 341

The table below provides examples of the settings used to connect to an Active Directory service and an OpenLDAP service to PE. Each setting is explained in more detail below the table.

GET /ds

Get the connected directory service information. Authentication is required.

**Return format**

Returns a 200 OK response with an object representing the connection.

For example:

```
{"display_name": "AD",
"hostname": "ds.foobar.com",
"port": 10379,
...
```

If the connection settings have not been specified, returns a 200 OK response with an empty JSON object.

For example:

```
{}
```

GET /ds/test

Runs a connection test for the connected directory service. Authentication is required.

**Return format**

If the connection test is successful, returns a 200 OK response with information about the test run.
For example:

```json
{"elapsed": 10}
```

**Error responses**

- 400 Bad Request if the request is malformed.
- 401 Unauthorized if no user is logged in.
- 403 Forbidden if the current user lacks the permissions to test the directory settings.

```json
{"elapsed": 20, "error": "..."}
```

**PUT /ds/test**

Performs a connection test with the submitted settings. Authentication is required.

**Request format**

Accepts the full set of directory settings keys with values defined.

**Return format**

If the connection test is successful, returns information about the test run.

For example:

```json
{"elapsed": 10}
```

**Error responses**

If the request times out or encounters an error, returns information about the test run.

For example:

```json
{"elapsed": 20, "error": "..."}
```

**PUT /ds**

Replaces current directory service connection settings. Authentication is required.

When changing directory service settings, you must specify all of the required directory service settings in the PUT request, including the required settings that are remaining the same. You do not need to specify optional settings unless you are changing them.

**Request format**

Accepts directory service connection settings. To "disconnect" the DS, PUT either an empty object ("{}") or the settings structure with all values set to null.

For example:

```json
{"hostname": "ds.somehost.com",
 "name"  
 "port": 10389,  
 "login": "frances", ...}
```

**Working with nested groups**

When authorizing users, the RBAC service has the ability to search nested groups. Nested groups are groups that are members of external directory groups. For example, if your external directory has a "System Administrators"...
group, and you’ve given that group the "Superusers" user role in RBAC, then RBAC also searches any groups that are members of the "System Administrators" group and assign the "Superusers" role to users in those member groups.

By default, RBAC does not search nested groups. To enable nested group searches, specify the `search_nested_groups` field and give it a value of `true`.

**Note:** In PE 2015.3 and earlier versions, RBAC’s default was to search nested groups. When upgrading from one of these earlier versions, this default setting is preserved and RBAC continues to search nested groups. If you have a large number of nested groups, you might experience a slowdown in performance when users are logging in because RBAC is searching the entire hierarchy of groups. To avoid performance issues, change the `search_nested_groups` field to `false` for a more shallow search in which RBAC searches only the groups it has been configured to use for user roles.

**Using StartTLS connections**

To use a StartTLS connection, specify "start_tls": true. When set to true, StartTLS is used to secure your connection to the directory service, and any certificates that you have configured through the DS trust chain setting is used to verify the identity of the directory service. When specifying StartTLS, make sure that you don’t also have SSL set to true.

**Disabling matching rule in chain**

When PE detects an Active Directory that supports the LDAP_MATCHING_RULE_IN_CHAIN feature, it automatically uses it. Under some specific circumstances, you might need to disable this setting. To disable it, specify "disable_ldap_matching_rule_in_chain": true in the PUT request. This is an optional setting.

**Return format**

Returns a 200 OK response with an object showing the updated connection settings.

For example:

```
{"hostname": "ds.somehost.com",
"port": 10389,
"login": "frances", ...}
```

**Password endpoints**

When local users forget passwords or lock themselves out of PE by attempting to log in with incorrect credentials too many times, you must generate a password reset token for them. The password endpoints enable you to generate password reset tokens for a specific local user or with a token that contains a temporary password in the body.

**Tip:** The PE console admin password can also be reset using a password reset script available on the PE console node.

**Tip:** 10 is the default number of login attempts that can be made with incorrect credentials before a user is locked out. You can change the value by configuring the failed-attempts-lockout parameter.

**Related information**

Reset the console administrator password on page 330

If you’re unable to log in to the console as admin, you can change the password from the command line of the node running console services.

**POST /users/:sid/password/reset**

Generates a single-use password reset token for the specified local user.

The generated token can be used to reset the password only one time and it has a limited lifetime. The lifetime is based on a configuration value

© 2022 Puppet, Inc., a Perforce company
puppet_enterprise::profile::console::rbac_password_reset_expiration (number of hours). The default value is 24 hours. Authentication is required.

This token is to be given to the appropriate user for use with the /auth/reset endpoint.

**Response format**

Returns a 201 Created response. For example:

```json
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhcGlfdXNlciIsInN1YiI6ImE3YzA4MTY3LWE1MDU0
0aUcdmBtuD95J9NGEAPtTwsDcux0_33hPz03m0Rp3hDSjb1QwyTBTuf2rpa
maYo1VRw-Eou60dUlqBAaqkQ9xrMtQ06Yqn0M6p9Rx1fk_u5YjZDrhLZsYxg-LY-
cY2IFow_XFIwc9vwWBuDXllwLB9TqJIo1ZNVNyomoHABzoKr
-v-dKGFRAWF-qBTw event-v-
doFSxlkBwkangGq2jb2Ghszhb8jdK9WW9O_8fnOcUK_kLlbE7W
0htK045tT9BPazwZaZWagHiojI_YfyJfBiB_d
F3XquRE-4C2EbejQLd_u-WZGnJp2V_HGK45pd56V-
anuDq1AakaW31zDJkJPuT4CUB8_xpFFLRdofTHrLPpwo5dsbKK-9x3JfIMhnsK4e2H_5Nywk0w951vonKuY
```

**Error responses**

Returns a 403 Forbidden response if the user does not have permissions to create a reset path for the specified user, or if the user is a remote user.

Returns a 404 Not Found response if a user with the given identifier does not exist.

**POST /auth/reset**

 Resets a local user's password using a one-time token obtained via the /users/:sid/password/reset endpoint, with the new password in the body. No authentication is required to use this endpoint.

**Request format**

The appropriate user is identified in the payload of the token. This endpoint does not establish a valid logged-in session for the user.

For example:

```json
{"token": "text of token goes here",
 "password": "someotherpassword"}
```

**Response format**

Returns a 200 OK response if the token is valid and the password has been successfully changed.

**Error responses**

Returns a 403 Permission Denied response if the token has already been used or is invalid.

**PUT /users/current/password**

Changes the password for the current local user. A payload containing the current password must be provided. Authentication is required.

**Request format**

The current and new passwords must both be included.

For example:

```json
{"current_password": "somepassword",
 "password": "someotherpassword"}
```
Response format
Returns a 200 OK response if the password has been successfully changed.

Error responses
Returns a 403 Forbidden response if the user is a remote user, or if current_password doesn't match the current password stored for the user. The body of the response includes a message that specifies the cause of the failure.

RBAC API v2
The role-based access control (RBAC) API v2 service enables you to validate groups and revoke tokens.

- Tokens endpoints on page 378
A user’s access to PE services can be controlled using authentication tokens. Users can revoke authentication tokens using the tokens endpoints.

Related information
API index on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

Tokens endpoints
A user’s access to PE services can be controlled using authentication tokens. Users can revoke authentication tokens using the tokens endpoints.

Related information
Token-based authentication on page 347
Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

DELETE /tokens
Use this endpoint to revoke one or more authentication tokens, ensuring the tokens can no longer be used with RBAC.

Parameters
The tokens must be specified using at least one of the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>revoke_tokens</td>
<td>A list of complete authentication tokens to be revoked.</td>
</tr>
<tr>
<td>revoke_tokens_by_usernames</td>
<td>A list of usernames whose tokens are to be revoked.</td>
</tr>
<tr>
<td>revoke_tokens_by_labels</td>
<td>A list of the labels of tokens to revoke (only tokens owned by the user making the request can be revoked in this manner).</td>
</tr>
</tbody>
</table>

You can supply parameters either as query parameters or as JSON-encoded in the body. If you include them as query parameters, separate the tokens, labels, or usernames by commas:

```
/tokens?revoke_tokens_by_usernames=<USER_NAME>,<USER_NAME>
```

If you encoded them in the body, supply them as a JSON array:

```
{"revoke_tokens_by_usernames": ["<USER_NAME>", "<USER_NAME>" ]}
```
If you provide a parameter as both a query parameter and in the JSON body, the values from the two sources are combined. It is not an error to specify the same token using multiple means (such as by providing the entire token to the `revoke_tokens` parameter and also including its label in the value of `revoke_tokens_by_labels`).

In the case of an error, malformed or otherwise input, or bad request data, the endpoint still attempts to revoke as many tokens as possible. This means it's possible to encounter multiple error conditions in a single request, such as if there were malformed usernames supplied in the request and a database error occurred when trying to revoke the well-formed usernames.

All operations on this endpoint are idempotent—it is not an error to revoke the same token two or more times.

Any user can revoke any token by supplying the complete token in the `revoke_tokens` query parameter or request body field.

To revoke tokens by username, the user making the request must have the "Users Revoke" permission for that user.

**Example JSON body**

The following is an example JSON body using each of the available parameters.

```json
{"revoke_tokens": [
  "eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhZG1pbiIsImIhdCI6MTQzOTQ5Mzg0NiwiXVJc
  tMGtAALp_vBONSQQ2zu5ihWfsDU4mmefbelR1CzA2TWrsv62DpJhtSc3iMLsSyjcujsP6187kwJw3XFTjcB79k
  aODfSvzf91351E7vi97s-9fFcHUPFzboyz60GDyRJVS0ohIWPnjM1AfX0UBitgqCYxne4rr8wKBoeOXoeEQezJ2u8-q0TgeN3ke4azDxdIqFhZ7H10-
  jDR0C5yeSBGWfX-0KEbp42cGz81A6rrIHpsaajRwUg9yThsUkT2crh6878orCLgfoBLDh-
  rOBTLie1al6sasH-ggpdHqVFTFOomEXM6UTJlp1NpuP01rNr9JM1xWhIBwpExH11__-136D1N14m32kw0-
  oV6GzXRx70xq_N2CwIwObw-X15aUUC4KkyPtDmNnvvC1n4"
]
"revoke_tokens_by_labels": ["Workstation Token", "VPS Token"],
"revoke_tokens_by_usernames": ["<USER NAME>", "<USER NAME>"]}
```

**Response codes**

The server uses the following response codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>No Content  Sent if all operations are successful.</td>
</tr>
<tr>
<td>500</td>
<td>Application Error  Sent if there was a database error when trying to revoke tokens.</td>
</tr>
<tr>
<td>403</td>
<td>Sent if the user lacks the permission to revoke one of the supplied usernames and no database error occurred.</td>
</tr>
<tr>
<td>Code</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>400 Malformed</td>
<td>Sent if one these conditions are true:</td>
</tr>
<tr>
<td></td>
<td>• At least one of the tokens, usernames, or labels is malformed.</td>
</tr>
<tr>
<td></td>
<td>• At least one of the usernames does not exist in the RBAC database.</td>
</tr>
<tr>
<td></td>
<td>• Neither <code>revoke_tokens</code>, <code>revoke_tokens_by_usernames</code>, nor</td>
</tr>
<tr>
<td></td>
<td><code>revoke_tokens_by_labels</code> is supplied.</td>
</tr>
<tr>
<td></td>
<td>• There are unrecognized query parameters or fields in the request body,</td>
</tr>
<tr>
<td></td>
<td>and the user has all necessary permissions and no database error occurred.</td>
</tr>
</tbody>
</table>

All error responses follow the standard JSON error format, meaning they have `kind`, `msg`, and `details` keys.

The `kind` key is `puppetlabs.rbac/database-token-error` if the response is a 500, `permission-denied` if the response is a 403, and `malformed-token-request` if the response is a 400.

The `msg` key contains an English-language description of the problems encountered while processing the request and performing the revocations, ending with either "No tokens were revoked" or "All other tokens were successfully revoked", depending on whether any operations were successful.

The `details` key contains an object with arrays in the `malformed_tokens`, `malformed_usernames`, `malformed_labels`, `nonexistent_usernames`, `permission_denied_usernames`, and `unrecognized_parameters` fields, as well as the Boolean field `other_tokens_revoked`.

The arrays all contain bad input from the request, and the `other_tokens_revoked` field's value indicates whether any of the revocations specified in the request were successful or not.

**Example error body**

The server returns an error body resembling the following:

```json
{"kind": "malformed-request",
 "msg": "The following user does not exist: FormerEmployee. All other tokens were successfully revoked.",
 "details": {
 "malformed_tokens": [],
 "malformed_usernames": [],
 "malformed_labels": [],
 "nonexistent_usernames": ["FormerEmployee"],
 "permission_denied_usernames": [],
 "unrecognized_parameters": [],
 "other_tokens_revoked": true}}
```

**DELETE /tokens/<token>**

Use this endpoint to revoke a single token, ensuring that it can no longer be used with RBAC. Authentication is required.

This endpoint is equivalent to `DELETE /tokens?revoke_tokens={TOKEN}`.

This API can be used only by the admin or API user.
Response codes

The server uses the following response codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>204 No Content</td>
<td>Sent if all operations are successful.</td>
</tr>
<tr>
<td>400 Malformed</td>
<td>Sent if the token is malformed.</td>
</tr>
</tbody>
</table>

The error response is identical to DELETE /tokens.

Example error body

The error response from this endpoint is identical to DELETE /tokens.

```
{"kind": "malformed-request",
 "msg": "The following token is malformed: notAToken. This can be caused by an error while copying and pasting. No tokens were revoked."
 "details": {"malformed_tokens": ["notAToken"],
 "malformed_labels": [],
 "malformed_usernames": [],
 "nonexistent_usernames": [],
 "permission_denied_usernames": [],
 "unrecognized_parameters": [],
 "other_tokens_revoked": false}}
```

POST /auth/token/authenticate

Use this endpoint to exchange a token for a map representing an RBAC subject and associated token data. This endpoint does not require authentication.

This endpoint accepts a JSON body containing entries for token and update_last_activity?. The token field is a string containing an authentication token. The update_last_activity? field is a Boolean data type that indicates whether a successful authentication should update the last_active timestamp for the token.

Response codes

The server uses the following response codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>The subject represented by the token.</td>
</tr>
<tr>
<td>400 invalid-token</td>
<td>The provided token was either tampered with or could not be parsed.</td>
</tr>
<tr>
<td>403 token-revoked</td>
<td>The provided token has been revoked.</td>
</tr>
<tr>
<td>403 token-expired</td>
<td>The token has expired and is no longer valid.</td>
</tr>
<tr>
<td>403 token-timed-out</td>
<td>The token has timed out due to inactivity.</td>
</tr>
</tbody>
</table>

Example JSON body

```
{
 "token": "0VZZ6geJQK8zJGKxBdg1atTsMLAsfCQFJuRwxsMNgCr4",
 "update_last_activity?:": false
}
```

Example return

```
{"description":null,
```
Activity service API

The activity service logs changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles.

- **Forming activity service API requests** on page 382
  Token-based authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or allowed certificates.

- **Event types reported by the activity service** on page 383
  Activity reporting provides a useful audit trail for actions that change role-based access control (RBAC) entities, such as users, directory groups, and user roles.

- **Events endpoints** on page 386
  Use the events endpoints to retrieve activity service events.

**Related information**

API index on page 32

APIS allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Forming activity service API requests**

Token-based authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or allowed certificates.

By default, the activity service listens on port 4433. All endpoints are relative to the /activity-api/ path. So, for example, the full URL for the /v1/events endpoint on localhost is https://localhost:4433/activity-api/v1/events.

**Authentication using tokens**

Insert a user authentication token variable in an activity service API request.

1. Generate a token: puppet-access login
2. Use puppet-access show to populate an authentication header, and use that header in your API request.

```shell
puppet-access login
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print server):4433/activity-api/v1/events?service_id=classifier"
```
Managing access

Examples of curl commands

See Usage notes for curl examples for information about forming curl commands.

Authenticating using an allowed certificate

You can also authenticate requests using a certificate listed in RBAC's certificate allowlist, located at /etc/puppetlabs/console-services/rbac-certificate-allowlist. Note that if you edit this file, you must reload the pe-console-services service (run sudo service pe-console-services reload) for your changes to take effect.

Attach the certificate using the command line, as demonstrated in the example curl query below. You must have the allowed certificate name (which must match a name in the /etc/puppetlabs/console-services/rbac-certificate-allowlist file) and the private key to run the script.

```bash
    type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/activity-api/v1/events?
    service_id=classifier"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
"$uri"
```

You do not need to use an agent certificate for authentication. You can use puppet cert generate to create a new certificate specifically for use with the API.

Related information

Token-based authentication on page 347

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Event types reported by the activity service

Activity reporting provides a useful audit trail for actions that change role-based access control (RBAC) entities, such as users, directory groups, and user roles.

Local users

These events are displayed in the console on the Activity tab for the affected user.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>A new local user is created. An initial value for each metadata field is</td>
<td>Created with login set to &quot;jean&quot;.</td>
</tr>
<tr>
<td>Metadata</td>
<td>Any change to the login, display name, or email keys.</td>
<td>Display name set to &quot;Jean Jackson&quot;.</td>
</tr>
<tr>
<td>Role membership</td>
<td>A user is added to or removed from a role. The display name and user ID of</td>
<td>User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cda) added to role</td>
</tr>
<tr>
<td>Authentication</td>
<td>The affected user are displayed.</td>
<td>Operators.</td>
</tr>
<tr>
<td></td>
<td>A user logs in. The display name and user ID of the affected user are</td>
<td>User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cda) logged in.</td>
</tr>
</tbody>
</table>
### Remote users

These events are displayed in the console on the **Activity** tab for the affected user.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role membership</td>
<td>A user is added to or removed from a role. These events are also shown on the page for the role. The display name and user ID of the affected user are displayed.</td>
<td>User Kalo Hill (76483e62-5ed4-11e4-aa15-123b93f75cba) added to role Viewers.</td>
</tr>
<tr>
<td>Revocation</td>
<td>A user is revoked or reinstated.</td>
<td>User revoked.</td>
</tr>
</tbody>
</table>

### Directory groups

These events are displayed in the console on the **Activity** tab for the affected group.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importation</td>
<td>A directory group is imported. The initial value for each metadata field is reported (these cannot be updated using the RBAC UI).</td>
<td>Created with display name set to &quot;Engineers&quot;.</td>
</tr>
<tr>
<td>Role membership</td>
<td>A group is added to or removed from a role. These events are also shown on the page for the role. The group's display name and ID are provided.</td>
<td>Group Engineers (7dee3acc-5ed4-11e4-aa15-123b93f75cba) added to role Operators.</td>
</tr>
</tbody>
</table>

### Roles

These events are displayed in the console on the **Activity** tab for the affected role.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>A role's display name or description changes.</td>
<td>Description set to &quot;Sysadmins with full privileges for node groups.&quot;</td>
</tr>
<tr>
<td>Event</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Members</td>
<td>A group is added to or removed from a role. The display name and ID of the user or group are provided. These events are also displayed on the page for the affected user or group.</td>
<td>User Kalo Hill (76483e62-5ed4-11e4-aa15-123b93f75cba) removed from role Operators.</td>
</tr>
<tr>
<td>Permissions</td>
<td>A permission is added to or removed from a role.</td>
<td>Permission users:edit:76483e62-5ed4-11e4-aa15-123b93f75cba added to role Operators.</td>
</tr>
<tr>
<td>Delete</td>
<td>A role has been removed.</td>
<td>The Delete event is recorded and available only through the activity service API, not the Activity tab.</td>
</tr>
</tbody>
</table>

### Orchestration

These events are displayed in the console on the Activity tab for the affected node.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent runs</td>
<td>Puppet runs as part of an orchestration job. This includes runs started from the orchestrator or the PE console.</td>
<td>Request Puppet agent run on node.example.com via orchestrator job 12.</td>
</tr>
<tr>
<td>Task runs</td>
<td>Tasks run as orchestration jobs set up in the console or on the command line.</td>
<td>Request echo task on neptune.example.com via orchestrator job 9,607</td>
</tr>
</tbody>
</table>

### Authentication tokens

These events are displayed in the console on the Activity tab on the affected user's page.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>A new token is generated. These events are exposed in the console on the Activity tab for the user who owns the token.</td>
<td>Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c) generated an authentication token.</td>
</tr>
<tr>
<td>Direct revocation</td>
<td>A successful token revocation request. These events are exposed in the console on the Activity tab for the user performing the revocation.</td>
<td>Administrator (42bf351c-f9ec-40af-84ad-e976f6c7f4bd) revoked an authentication token belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c), issued at 2016-02-17T21:53:23.000Z and expiring at 2016-02-17T21:58:23.000Z.</td>
</tr>
</tbody>
</table>
### Revocation by username

All tokens for a username are revoked. These events are exposed in the console on the Activity tab for the user performing the revocation.

Example:

Administrator (42bf351cf9ec-40af-84ad-e976f6c7f4bd) revoked all authentication tokens belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c).

### Directory service settings

These events are not exposed in the console. The activity service API must be used to see these events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update settings (except password)</td>
<td>A setting is changed in the directory service settings.</td>
<td>User rdn set to &quot;ou=users&quot;.</td>
</tr>
<tr>
<td>Update directory service password</td>
<td>The directory service password is changed.</td>
<td>Password updated.</td>
</tr>
</tbody>
</table>

### Events endpoints

Use the events endpoints to retrieve activity service events.

**GET /v1/events**

Fetches activity service events. Web session authentication is required.

**Request format**

The /v1/events endpoint supports filtering through query parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>The ID of the service. Required.</td>
</tr>
<tr>
<td>subject_type</td>
<td>The subject who performed the activity. Required only when subject_id is provided.</td>
</tr>
<tr>
<td>subject_id</td>
<td>Comma-separated list of IDs associated with the defined subject type. Optional.</td>
</tr>
<tr>
<td>object_type</td>
<td>The object affected by the activity. Required only when object_id is provided.</td>
</tr>
<tr>
<td>object_id</td>
<td>Comma-separated list of IDs associated with the defined object type. Optional.</td>
</tr>
<tr>
<td>offset</td>
<td>Number of commits to skip before returning events. Optional.</td>
</tr>
<tr>
<td>order</td>
<td>Specifies the sorting order of the events, in submit time. One of either asc or desc. Defaults to desc.</td>
</tr>
<tr>
<td>limit</td>
<td>Maximum number of events to return. Default is 1000 events. Optional.</td>
</tr>
<tr>
<td>after_service_commit_time</td>
<td>Return results after a specific service commit time. ISO 8601. Optional.</td>
</tr>
</tbody>
</table>
**Response format**

Responses are returned in a structured JSON format.

GET /v1/events?service_id=classifier&subject_type=users&subject_id=kai

```
{
  "commits": [
    {
      "object": {
        "id": "415dfsdf-dfgd45dfg-4dsf54d",
        "name": "Default Node Group"
      },
      "subject": {
        "id": "dfgdfc145-545dfg5f-fdg45s55",
        "name": "Kai Evans"
      },
      "timestamp": "2014-06-24T04:00:00Z",
      "events": [
        {
          "message": "Create Node"
        },
        {
          "message": "Create Node Class"
        }
      ]
    }
  ],
  "total-rows": 1
}
```

GET /v1/events?service_id=classifier&object_type=node_groups&object_id=2

```
{
  "commits": [
    {
      "object": {
        "id": "415dfsdf-dfgd45dfg-4dsf54d",
        "name": "Default Node Group"
      },
      "subject": {
        "id": "dfgdfc145-545dfg5f-fdg45s55",
        "name": "Kai Evans"
      },
      "timestamp": "2014-06-24T04:00:00Z",
      "events": [
        {
          "message": "Create Node"
        },
        {
          "message": "Create Node Class"
        }
      ]
    }
  ],
  "total-rows": 1
}
```
GET /v1/events.csv
Fetched activity service events and returns in a flat CSV format. Token-based authentication is required.

Request format

The /v1/events.csv endpoint supports accepts the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>The ID of the service. Required.</td>
</tr>
<tr>
<td>subject_type</td>
<td>The subject who performed the activity. Required only when subject_id is provided.</td>
</tr>
<tr>
<td>subject_id</td>
<td>Command-separated list of IDs associated with the defined subject type. Optional.</td>
</tr>
<tr>
<td>object_type</td>
<td>The object affected by the activity. Required only when object_id is provided.</td>
</tr>
<tr>
<td>object_id</td>
<td>Comma-separated list of IDs associated with the defined object type. Optional.</td>
</tr>
<tr>
<td>offset</td>
<td>Number of commits to skip before returning events. Optional.</td>
</tr>
<tr>
<td>order</td>
<td>Specifies the sorting order of the events, in submit time. One of either asc or desc. Defaults to desc.</td>
</tr>
<tr>
<td>limit</td>
<td>Maximum number of events to return. Default is 10000 events. Optional.</td>
</tr>
</tbody>
</table>

Response format

Responses are returned in a flat CSV format. Example return:

```
Submit Time,Subject Type,Subject Id,Subject Name,Object Type,Object Id,Object Name,Type,What,Description,Message
YYYY-MM-DD 18:52:27.76,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,delete,node_group,delete_node_group,"Deleted the ""web_servers"" group with id b55c209d-e68f-4096-9a2c-5ae52dd2500c"
YYYY-MM-DD 18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,create,node_group,create_node_group,"Created the ""web_servers"" group with id b55c209d-e68f-4096-9a2c-5ae52dd2500c"
YYYY-MM-DD 18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_description,edit_node_group_description,"Changed the description to ""
YYYY-MM-DD 18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_environment,edit_node_group_environment,"Changed the environment to "production"
YYYY-MM-DD 18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_environment_override,edit_node_group_environment_override,"Changed the environment override setting to false"
YYYY-MM-DD 18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_parent,edit_node_group_parent,"Changed the parent to ec519937-8681-43d3-8b74-380d65736dba"
```
GET /v2/events
Feteches events in a structured JSON format. Allows filtering through query parameters and supports multiple objects for filtering results. Requires token based authentication.

Request format
The /v2/events endpoint allows multiple optional objects for filtering results. You can specify as many parameters as you want to filter results. Filtering uses "or" logic, except for the timestamp ranges which use "and" logic.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>The ID of the service. Optional.</td>
</tr>
<tr>
<td>offset</td>
<td>Number of commits to skip before returning events. Optional.</td>
</tr>
<tr>
<td>order</td>
<td>Specifies the sorting order of the events, in submit time. One of either asc or desc. Defaults to desc.</td>
</tr>
<tr>
<td>limit</td>
<td>Maximum number of events to return. Default is 1000 events. Optional.</td>
</tr>
<tr>
<td>query</td>
<td>A JSON-encoded array of optional objects used to filter the results. You can specify multiple objects of the same or different types with the exception of ip_address. You can only specify one ip_address. Results returned based on this logic: (and (or , , ) (or )))]. Optional.</td>
</tr>
</tbody>
</table>

The query parameter supports one or more of following:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject_id</td>
<td>ID associated with the defined subject type. Must be paired with subject_type. Optional.</td>
</tr>
<tr>
<td>subject_type</td>
<td>The subject who performed the activity. Defaults to users. Required only when subject_id is provided.</td>
</tr>
<tr>
<td>object_id</td>
<td>ID associated with the defined object type. If used, it must be paired with object_type. Optional.</td>
</tr>
<tr>
<td>object_type</td>
<td>The object affected by the activity. Optional.</td>
</tr>
<tr>
<td>ip_address</td>
<td>Specifies the ip address. Partial string match. Optional.</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>ISO 8601 timestamp for start time. Uses &quot;and&quot; logic and must be paired with end. Optional.</td>
</tr>
<tr>
<td>end</td>
<td>ISO 8601 timestamp for end time. Uses &quot;and&quot; logic and must be paired with start. Optional.</td>
</tr>
</tbody>
</table>

For example, this curl request returns all events performed by users 42bf351c-f9ec-40af-84ad-e976feca7f4bd and 95c6159e-dfa3-4f34-b884-45e2ba39d24b from 01 November 2019 through 01 December 2019.

```bash
curl --insecure --request GET \
--header "X-Authentication: $(puppet-access show)" \
-G https://$(puppet config print server):4433/activity-api/v2/events" \
--data-urlencode 'service_id=classifier' \
--data-urlencode 'query=["object_id": "db2caca1-d6a4-4145-8240-9de9b4e654d1","object_type": "users"], \
{"subject_id": "db2caca1-d6a4-4145-8240-9de9b4e654d1"}, \
{"object_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8", \
"object_type": "users"}, \
{"subject_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8"}, \
{"start": "2019-11-01T21:32:39Z", "end": "2019-12-01T00:00:00Z"}]
```

See Usage notes for curl examples for information about forming curl commands.

**Response format**

Responses are returned in a structured JSON format.

For example, the above request produces the following response:

```json
{
  "commits": [
    {
      "objects": [
        {
          "id": "5a359d65-a8e5-41f5-b99d-3f0e6b5d668d",
          "name": "PE Agent",
          "type": "node_groups"
        },
        {
          "id": "db2caca1-d6a4-4145-8240-9de9b4e654d1",
          "name": "kai.evans"
        },
        {
          "timestamp": "2019-11-19T16:40:12Z",
          "events": [
            {
              "message": "Added the \"package_inventory_enabled\" parameter to the \"puppet Enterprise::profile::agent\" class",
              "type": "edit",
              "what": "node_group_class_parameter",
              "description": "add_node_group_class_parameter_puppet_enterprise::profile::agent_package_inventory_enabled"
            }
          ]
        }
      ],
      "subject": {
        "id": "db2caca1-d6a4-4145-8240-9de9b4e654d1",
        "name": "kai.evans"
      },
      "timestamp": "2019-11-19T16:40:12Z",
      "events": [
        {
          "message": "Added the \"package_inventory_enabled\" parameter to the \"puppet Enterprise::profile::agent\" class",
          "type": "edit",
          "what": "node_group_class_parameter",
          "description": "add_node_group_class_parameter_puppet_enterprise::profile::agent_package_inventory_enabled"
        }
      ]
    }
  ],
  "objects": [
    {
      "id": "6977ec72-5be3-4e0e-975e-a8d144b9f7ea",
```
Other examples

Request:

```bash
curl --insecure --request GET
--header "X-Authentication: $(puppet-access show)"
-G "https://$(puppet config print server):4433/activity-api/v2/events"
--data-urlencode 'service_id=classifier'
--data-urlencode 'query=[{"subject_id": "dfgdfc145-545dfg54f-fdg45s5s",
"subject_type": "users"},
{"object_id": "415dfsvdf-dfgd45dfg-4dsfg54d", "object_type": "node_group"}]'
```

Response:

```json
{
 "commits": [
 {"objects": [{"id": "415dfsvdf-dfgd45dfg-4dsfg54d",
 "name": "Default Node Group" }]
 },
 "subject": {
 "id": "dfgdfc145-545dfg54f-fdg45s5s",
 "name": "Kai Evans"
 },
 "timestamp": "2014-06-24T04:00:00Z",
 "events": [
 {"message": "Create Node"
 },
 {"message": "Create Node Class"
 }
 ],
 "pagination": {"total": "1", "limit": "1000", "offset": "0"}
}
```

Request:

```bash
curl --insecure --request GET \
```

© 2022 Puppet, Inc., a Perforce company
GET /v2/events.csv
Fetched events in a flat CSV format. Allows filtering through query parameters and supports multiple objects for filtering results. Requires token based authentication.

Request format
The /v2/events.csv endpoint allows multiple optional objects for filtering results. You can specify as many parameters as you want to filter results. Filtering uses "or" logic, except for the timestamp ranges which use "and" logic.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>The ID of the service. Optional.</td>
</tr>
<tr>
<td>offset</td>
<td>Number of commits to skip before returning events. Optional.</td>
</tr>
<tr>
<td>order</td>
<td>Specifies the sorting order of the events, in submit time. One of either asc or desc. Defaults to desc.</td>
</tr>
<tr>
<td>limit</td>
<td>Maximum number of events to return. Default is 1000 events. Optional.</td>
</tr>
<tr>
<td>query</td>
<td>A JSON-encoded array of optional objects used to filter the results. Multiple objects of the same or different types can be specified. Results returned based on this logic: (and (or . . .) (or . . .)). Optional.</td>
</tr>
</tbody>
</table>

The query parameter supports one or more of following:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject_id</td>
<td>ID associated with the defined subject type. Must be paired with subject_type. Optional.</td>
</tr>
<tr>
<td>subject_type</td>
<td>The subject who performed the activity. Defaults to users. Required only when subject_id is provided.</td>
</tr>
<tr>
<td>object_id</td>
<td>ID associated with the defined object type. Must be paired with object_type. Optional.</td>
</tr>
<tr>
<td>object_type</td>
<td>The object affected by the activity. Required only when the object_id is provided.</td>
</tr>
<tr>
<td>ip_address</td>
<td>Specifies the ip address. Optional.</td>
</tr>
<tr>
<td>start</td>
<td>ISO 8601 timestamp for start time. Uses &quot;and&quot; logic and must be paired with end. Optional.</td>
</tr>
<tr>
<td>end</td>
<td>ISO 8601 timestamp for end time. Uses &quot;and&quot; logic and must be paired with start. Optional.</td>
</tr>
</tbody>
</table>

**Response format**

GET `/v2/events.csv?service_id=classifier&query=[{"subject_id": "kai", "subject_type": "users"}]`

<table>
<thead>
<tr>
<th>Submit Time,Subject Type,Subject Id,Subject Name,Object Type,Object Id,Object Name,Type,What,Description,Message,Ip Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-07-17 13:08:09.985221,users,kai,Kai Evans,node_groups,2,Default Node Group,create,node,create_node,Create Node,123.123.123.123</td>
</tr>
<tr>
<td>2014-07-17 13:08:09.985221,users,kai,Kai Evans,node_groups,2,Default Node Group,create,node_class,create_node_class,Create Node Class,123.123.123.123</td>
</tr>
</tbody>
</table>

**Monitoring and reporting**

The console offers a variety of tools you can use to monitor the current state of your infrastructure, see the results of planned or unplanned changes to your Puppet code, view reports, and investigate problems. These tools are grouped in the Enforcement section of the console’s sidebar.

- **Monitoring infrastructure state** on page 396
  
  When nodes fetch their configurations from the primary server, they send back inventory data and a report of their run. This information is summarized on the Status page in the console.

- **Viewing and managing packages** on page 401
  
  The Packages page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the Packages page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals. Package management is available for all agent nodes.

© 2022 Puppet, Inc., a Perforce company
• **Value report** on page 403
  Value analytics give you insight into time and money freed by PE automation. You can access these analytics by viewing the **Value report** page in the console or using the value API.

• **Infrastructure reports** on page 408
  Each time Puppet runs on a node, it generates a report that provides information such as when the run took place, any issues encountered during the run, and the activity of resources on the node. These reports are collected on the **Reports** page in the console.

• **Analyzing changes across Puppet runs** on page 411
  The **Events** page in the console shows a summary of activity in your infrastructure. You can analyze the details of important changes, and investigate common causes behind related events. You can also examine specific class, node, and resource events, and find out what caused them to fail, change, or run as no-op.

• **Puppet Enterprise metrics** on page 414
  You can use Puppet Enterprise (PE) metrics for your own performance tuning or provide them to Support for troubleshooting.

• **Viewing and managing Puppet Server metrics** on page 415
  Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

• **Sample Grafana dashboard JSON code** on page 423
  The following JSON code is an example metrics dashboard. Copy and save it to a .json file to reuse it.

• **Status API** on page 436
  The status API allows you to check the health of PE components and services. It can be useful for automated monitoring of your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

---

**Monitoring infrastructure state**

When nodes fetch their configurations from the primary server, they send back inventory data and a report of their run. This information is summarized on the **Status** page in the console.

The **Status** page displays the most recent run status of each of your nodes so you can quickly find issues and diagnose their causes. You can also use this page to gather essential information about your infrastructure at a glance, such as how many nodes your primary server is managing, and whether any nodes are unresponsive.

**Node run statuses**

The **Status** page displays the run status of each node following the most recent Puppet run. There are 10 possible run statuses.

**Nodes run in enforcement mode**

⚠️ **With failures**

This node’s last Puppet run failed, or Puppet encountered an error that prevented it from making changes.

The error is usually tied to a particular resource (such as a file) managed by Puppet on the node. The node as a whole might still be functioning normally. Alternatively, the problem might be caused by a situation on the primary server, preventing the node’s agent from verifying whether the node is compliant.

⚠️ **With corrective changes**

During the last Puppet run, Puppet found inconsistencies between the last applied catalog and this node’s configuration, and corrected those inconsistencies to match the catalog.

**Note:** Corrective change reporting is available only on agent nodes running PE 2016.4 and later. Agents running earlier versions report all change events as "with intentional changes."
With intentional changes
During the last Puppet run, changes to the catalog were successfully applied to the node.

Unchanged
This node's last Puppet run was successful, and it was fully compliant. No changes were necessary.

Nodes run in no-op mode

Note: No-op mode reporting is available only on agent nodes running PE 2016.4 and later. Agents running earlier versions report all no-op mode runs as "would be unchanged."

With failures
This node’s last Puppet run in no-op mode failed, or Puppet encountered an error that prevented it from simulating changes.

Would have corrective changes
During the last Puppet run, Puppet found inconsistencies between the last applied catalog and this node’s configuration, and would have corrected those inconsistencies to match the catalog.

Would have intentional changes
During the last Puppet run, catalog changes would have been applied to the node.

Would be unchanged
This node’s last Puppet run was successful, and the node was fully compliant. No changes would have been necessary.

Nodes not reporting

Unresponsive
The node hasn't reported to the primary server recently. Something might be wrong. The cutoff for considering a node unresponsive defaults to one hour, and can be configured via the puppet Enterprise::console_services::no_longer_reporting_cutoff parameter. See Configure the PE console and console-services on page 299 for more information.

Check the run status table to see the timestamp for the last known Puppet run for the node and an indication of whether its last known run was in no-op mode. Correct the problem to resume Puppet runs on the node.

Have no reports
Although Puppet Server is aware of this node's existence, the node has never submitted a Puppet report for one or more of the following reasons: it's a newly commissioned node; it has never come online; or its copy of Puppet is not configured correctly.

Note: Expired or deactivated nodes are displayed on the Status page for seven days. To extend the amount of time that you can view or search for these nodes, change the node-ttl setting in PuppetDB. Changing this setting affects resources and exported resources.
Special categories

In addition to reporting the run status of each node, the **Status** page provides a secondary count of nodes that fall into special categories.

**Intended catalog failed**

During the last Puppet run, the intended catalog for this node failed, so Puppet substituted a cached catalog, as per your configuration settings.

This typically occurs when you have compilation errors in your Puppet code. Check the Puppet run's log for details.

This category is shown only if one or more agents fail to retrieve a valid catalog from Puppet Server.

**Enforced resources found**

During the last Puppet run in no-op mode, one or more resources was enforced, as per your use of the `noop => false` metaparameter setting.

This category is shown only if enforced resources are present on one or more nodes.

**How Puppet determines node run statuses**

Puppet uses a hierarchical system to determine a single run status for each node. This system gives higher priority to the activity types most likely to cause problems in your deployment, so you can focus on the nodes and events most in need of attention.

During a Puppet run, several activity types might occur on a single node. A node's run status reflects the activity with the highest alert level, regardless of how many events of each type took place during the run. Failure events receive the highest alert level, and no change events receive the lowest.

<table>
<thead>
<tr>
<th>Run status</th>
<th>Definitely happened</th>
<th>Might also have happened</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Failure" /></td>
<td>Failure</td>
<td>Corrective change, intentional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>change, no change</td>
</tr>
<tr>
<td><img src="image" alt="Corrective change" /></td>
<td>Corrective change</td>
<td>Intentional change, no change</td>
</tr>
<tr>
<td><img src="image" alt="Intentional change" /></td>
<td>Intentional change</td>
<td>No change</td>
</tr>
<tr>
<td><img src="image" alt="No change" /></td>
<td>No change</td>
<td></td>
</tr>
</tbody>
</table>

For example, during a Puppet run in enforcement mode, a node with 100 resources receives intentional changes on 30 resources, corrective changes on 10 resources, and no changes on the remaining 60 resources. This node's run status is "with corrective changes."

Node run statuses also prioritize run mode (either enforcement or no-op) over the state of individual resources. This means that a node run in no-op mode is always reported in the **Nodes run in no-op** column, even if some of its resource changes were enforced. Suppose the no-op flags on a node's resources are all set to false. Changes to the resources are enforced, not simulated. Even so, because it is run in no-op mode, the node's run status is "would have intentional changes."
Filtering nodes on the Status page

You can filter the list of nodes displayed on the Status page by run status and by node fact. If you set a run status filter, and also set a node fact filter, the table takes both filters into account, and shows only those nodes matching both filters.

Clicking Remove filter removes all filters currently in effect.

The filters you set are persistent. If you set run status or fact filters on the Status page, they continue to be applied to the table until they're changed or removed, even if you navigate to other pages in the console or log out. The persistent storage is associated with the browser tab, not your user account, and is cleared when you close the tab.

**Important:** The filter results count and the fact filter matching nodes counts are cached for two minutes after first retrieval. This reduces the total load on PuppetDB and decreases page load time, especially for fact filters with multiple rows. As a result, the displayed counts might be up to two minutes out of date.

Filter by node run status

The status counts section at the top of the Status page shows a summary of the number of nodes with each run status as of the last Puppet run. Filter nodes by run status to quickly focus on nodes with failures or change events.

In the status counts section, select a run status (such as with corrective changes or have no reports) or a run status category (such as Nodes run in no-op).

Filter by node fact

You can create a highly specific list of nodes for further investigation by using the fact filter tool.

For example, you can check that nodes you've updated have successfully changed, or find out the operating systems or IP addresses of a set of failed nodes to better understand the failure. You might also filter by facts to fulfill an auditor's request for information, such as the number of nodes running a particular version of software.

1. Click Filter by fact value. In the Fact field, select one of the available facts. An empty fact field is not allowed.

   **Tip:** To see the facts and values reported by a node on its most recent run, click the node name in the Run status table, then select the node’s Facts tab.

2. Select an Operator:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>is</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>is not</td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>matches a regular expression</td>
<td>Select this operator to use wildcards and other regular expressions if you want to find matching facts without having to specify the exact value.</td>
</tr>
<tr>
<td>~</td>
<td>(regex)</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>does not match a regular expression</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
</tbody>
</table>
3. In the **Value** field, enter a value. Strings are case-sensitive, so make sure you use the correct case.

The filter displays an error if you use an invalid string operator (for example, selecting a numeric value operator such as `>=` and entering a non-numeric string such as `pilsen` as the value) or enter an invalid regular expression.

**Note:** If you enter an invalid or empty value in the **Value** field, PE takes the following action in order to avoid a filter error:
- Invalid or empty Boolean facts are processed as **false**, and results are retrieved accordingly.
- Invalid or empty numeric facts are processed as 0, and results are retrieved accordingly.
- Invalid or incomplete regular expressions invalidate the filter, and no results are retrieved.

4. Click **Add**.

5. As needed, repeat these steps to add additional filters. If filtering by more than one node fact, specify either **Nodes must match all rules** or **Nodes can match any rule**.

**Filtering nodes in your node list**

Filter your node list by node name or by PQL query to more easily inspect them.

**Filter your node list by node name**

Filter your nodes list by node name to inspect them as a group.

Select **Node name**, type in the word you want to filter by, and click **Submit**.

**Filter your nodes by PQL query**

Filter your nodes list using a common PQL query.

Filtering your nodes list by PQL query enables you to manage them by specific factors, such as by operating system, report status, or class.

Specify a target by doing one of the following:
- Enter a query that selects the target you want. See the Puppet Query Language (PQL) reference for more information.
- Click **Common queries**. Select one of the queries and replace the defaults in the braces (`{ }`) with values that specify the target you want.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes</td>
<td><code>nodes[certname] { }</code></td>
</tr>
<tr>
<td>Nodes with a specific resource (example: httpd)</td>
<td><code>resources[certname] { type = &quot;Service&quot; and title = &quot;httpd&quot; }</code></td>
</tr>
<tr>
<td>Nodes with a specific fact and value (example: OS name is CentOS)</td>
<td><code>inventory[certname] { facts.os.name = &quot;&lt;OS&gt;&quot; }</code></td>
</tr>
<tr>
<td>Nodes with a specific report status (example: last run failed)</td>
<td><code>reports[certname] { latest_report_status = &quot;failed&quot; }</code></td>
</tr>
<tr>
<td>Nodes with a specific class (example: Apache)</td>
<td><code>resources[certname] { type = &quot;Class&quot; and title = &quot;Apache&quot; }</code></td>
</tr>
<tr>
<td>Nodes assigned to a specific environment (example: production)</td>
<td><code>nodes[certname] { catalog_environment = &quot;production&quot; }</code></td>
</tr>
<tr>
<td>Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)</td>
<td><code>resources[certname] { type = &quot;Package&quot; and title=&quot;openssl&quot; and parameters.ensure = &quot;1.0.1e-51.el7_2.7&quot; }</code></td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
Monitor PE services

PE includes console and command line tools for monitoring the status of core services.

<table>
<thead>
<tr>
<th>Component or service</th>
<th>Status monitor</th>
<th>Status command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity service</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Agentless Catalog Executor (ACE) service</td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>Bolt service</td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>Classifier service</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Code Manager service</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Orchestrator service</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Puppet Communications Protocol (PCP) broker</td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Role-based access control (RBAC) service</td>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>

**View the Puppet Services status monitor**

The **Puppet Services status** monitor provides a visual overview of the current state of core services, and can be used to quickly determine whether an unresponsive or restarting service is causing an issue with your deployment.

A checkmark appears next to **Puppet Services status** if all applicable services are accepting requests. In the event that no data is available, a question mark appears next to the link. If one or more services is restarting or not accepting requests, a warning icon appears.

1. In the console, click **Status**.
2. Click **Puppet Services status** to open the monitor.

**puppet infrastructure status command**

The **puppet infrastructure status command** displays errors and alerts from PE components and services. The command reports separately on the primary server and any compilers or replicas in your environment. You must run the command as root.

**Viewing and managing packages**

The **Packages** page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the **Packages** page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals. Package management is available for all agent nodes.
Enable package data collection

Package data collection is disabled by default, so the Packages page in the console initially appears blank. In order to view a node's current package inventory, enable package data collection.

You can choose to collect package data on all your nodes, or just a subset. Package inventory reporting is available on nodes with Puppet agent 1.6.0 or later installed, including nodes that don't have an active configuration on the Puppet Server.

1. In the console, click Node groups.
   - If you want to collect package data on all your nodes, click the PE Agent node group.
   - If you want to collect package data on a subset of your nodes, click Add group and create a new classification node group. Select PE Agent as the group's parent name. After the new node group is set up, use the Rules tab to dynamically add the relevant nodes.

2. Click Classes. In the Add new class field, select puppet_enterprise::profile::agent and click Add class.

3. In the puppet_enterprise::profile::agent class, set the Parameter to package_inventory_enabled and the Value to true. Click Add parameter, and commit changes.

4. Run Puppet to apply these changes to the nodes in your node group.
   - Puppet enables package inventory collection on this Puppet run, and begins collecting package data and reporting it on the Packages page on each subsequent Puppet run.

5. Run Puppet a second time to begin collecting package data, then click Packages.

View and manage package inventory

To view and manage the complete inventory of packages on your systems, use the Packages page in the console.

Before you begin

Make sure you have enabled package data collection for the nodes you wish to view.

**Tip:** If all the nodes on which a certain package is installed are deactivated, but the nodes' specified node-purge-ttl period has not yet elapsed, instances of the package still appear in summary counts on the Packages page. To correct this issue, adjust the node-purge-ttl setting and run garbage collection.

1. Run Puppet to collect the latest package data from your nodes.
2. In the console, click Packages to view your package inventory. To narrow the list of packages, enter the name or partial name of a package in the Filter by package name field and click Apply.
3. Click any package name or version to enter the detail page for that package.
4. On a package's detail page, use the Version selector to locate nodes with a particular package version installed.
5. Use the Instances selector to locate nodes where the package is not managed with Puppet, or to view nodes on which a package instance is managed with Puppet.

   To quickly find the place in your manifest where a Puppet-managed package is declared, select a code path in the Instances selector and click Copy path.

6. To modify a package on a group of nodes:
   - If the package is managed with Puppet, select a code path in the Instances selector and click Copy path, then navigate to and update the manifest.
   - If the package is not managed with Puppet, click Run > Task and create a new task.
View package data collection metadata

The puppet_inventory_metadata fact reports whether package data collection is enabled on a node, and shows the time spent collecting package data on the node during the last Puppet run.

Before you begin

Make sure you have enabled package data collection for the nodes you wish to view.

1. Click Node groups and select the node group you created when enabling package data collection.
2. Click Matching nodes and select a node from the list.
3. On the node's inventory page, click Facts and locate puppet_inventory_metadata in the list.

   The fact value looks something like:

   ```json
   {
     "packages" : {
       "collection_enabled" : true,
       "last_collection_time" : "1.9149s"
     }
   }
   ```

Disable package data collection

If you need to disable package data collection, set package_inventory_enabled to false and run Puppet twice.

1. Click Node groups and select the node group you used when enabling package data collection.
2. On the Classes tab, find the puppet_enterprise:profile::agent class, locate package_inventory_enabled parameter, and click Edit.
3. Change the Value of package_inventory_enabled to false, then commit changes.
4. Run Puppet to apply these changes to the nodes in your node group and disable package data collection.

   Package data is collected for a final time during this run.
5. Run Puppet a second time to purge package data from the impacted nodes' storage.

Value report

Value analytics give you insight into time and money freed by PE automation. You can access these analytics by viewing the Value report page in the console or using the value API.

The value report and the value API provide details about automated changes that PE makes to nodes, and provides an estimate of time freed by each type of change based on intelligent defaults or values you provide. You can also specify an average hourly salary and see an estimate of cost savings for all automated changes.

Follow these best practices to ensure your value analysis is accurate:

- Failed runs aren't tallied by value analytics, so ensure that tasks, plans, and Puppet runs are processing normally.
- If you are using the API, query the endpoint on a regular basis to gather data over a period of time.

Value report defaults

The value report provides an estimate of time freed by automated changes based on intelligent defaults. These defaults are based on customer research and take into account time to triage, research, and fix issues, as well as context switching.

When you query the value API, you can specify low, med, or high estimates for time freed parameters, or provide an exact value in minutes based on averages in your business. In the console value report, you specify an exact value in minutes. Unless you select otherwise, both the API and the console value report use the med value to estimate time freed by automation.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minutesFreedPerCorrectiveChange</td>
<td>• low (90 minutes)</td>
</tr>
<tr>
<td></td>
<td>• med (180 minutes)</td>
</tr>
<tr>
<td></td>
<td>• high (360 minutes)</td>
</tr>
<tr>
<td>minutesFreedPerIntentionalChange</td>
<td>• low (30 min)</td>
</tr>
<tr>
<td></td>
<td>• med (90 minutes)</td>
</tr>
<tr>
<td></td>
<td>• high (180 minutes)</td>
</tr>
<tr>
<td>minutesFreedPerTaskRun</td>
<td>• low (30 minutes)</td>
</tr>
<tr>
<td></td>
<td>• med (90 minutes)</td>
</tr>
<tr>
<td></td>
<td>• high (180 minutes)</td>
</tr>
<tr>
<td>minutesFreedPerPlanRun</td>
<td>• low (90 minutes)</td>
</tr>
<tr>
<td></td>
<td>• med (180 minutes)</td>
</tr>
<tr>
<td></td>
<td>• high (360 minutes)</td>
</tr>
</tbody>
</table>

You can change the low, med, and high estimates of time freed by specifying any of the value_report_* parameters in the PE Console node group in the puppet_enterprise::profile::console class.

GET /api/reports/value
Use the /value endpoint to retrieve information about time and money freed by PE automation.

Forming requests
Requests to the value API must be well-formed HTTP(S) requests.

By default, the value API uses the standard HTTPS port for console communication, port 443. You can omit the port from requests unless you want to specify a different port.

Authenticating
You must authenticate requests to the value API using your Puppet CA certificate and an RBAC token. The RBAC token must have viewing permissions for the console.

A simple, authenticated API request without any parameters looks like this:

```bash
curl -X GET --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
-H "X-Authentication: <RBAC_TOKEN>" \
"https://<HOSTNAME>/api/reports/value"
```

See Usage notes for curl examples for information about forming curl commands.

Query parameters
The request accepts the following parameters, appended with --data-urlencode:
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>averageHourlySalary</td>
<td>Numeric value specifying average hourly cost savings for automated work.</td>
<td>none</td>
</tr>
<tr>
<td>startDate</td>
<td>Date in yyyy-mm-dd format.</td>
<td>1 week ago +2 days</td>
</tr>
<tr>
<td>endDate</td>
<td>Date in yyyy-mm-dd format.</td>
<td>today -2 days</td>
</tr>
<tr>
<td>minutesFreedPerCorrectiveChange</td>
<td>low, med, high, any numeric value in minutes</td>
<td>med</td>
</tr>
<tr>
<td>minutesFreedPerIntentionalChange</td>
<td>low, med, high, any numeric value in minutes</td>
<td>med</td>
</tr>
<tr>
<td>minutesFreedPerTaskRun</td>
<td>low, med, high, any numeric value in minutes</td>
<td>med</td>
</tr>
<tr>
<td>minutesFreedPerPlanRun</td>
<td>low, med, high, any numeric value in minutes</td>
<td>med</td>
</tr>
</tbody>
</table>

### Response format

The response is a JSON object that lists details about time and cost freed, using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDate</td>
<td>Start date for the reporting period.</td>
</tr>
<tr>
<td>endDate</td>
<td>End date for the reporting period.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>totalCorrectiveChanges</td>
<td>Total number of corrective changes made during the reporting period.</td>
</tr>
<tr>
<td>minutesFreedByCorrectiveChanges</td>
<td>Total number of minutes freed by automated changes that prevent drift during regular Puppet runs. The calculation is based on the average minutes saved per change, as specified by the minutesFreedPerCorrectiveChange query parameter.</td>
</tr>
<tr>
<td>totalIntentionalChanges</td>
<td>Total number of intentional changes made during the reporting period.</td>
</tr>
<tr>
<td>minutesFreedByIntentionalChanges</td>
<td>Total number of minutes freed by automated changes based on new values or Puppet code. This calculation is based on the average minutes saved per change, as specified by the minutesFreedPerIntentionalChange query parameter.</td>
</tr>
<tr>
<td>totalNodesAffectedByTaskRuns</td>
<td>Total number of nodes affected by successful task runs during the reporting period.</td>
</tr>
<tr>
<td>minutesFreedByTaskRuns</td>
<td>Total number of minutes freed by automated task runs. This calculation is based on the average minutes saved per task run, as specified by the minutesFreedPerTaskRun query parameter.</td>
</tr>
<tr>
<td>totalNodesAffectedByPlanRuns</td>
<td>Total number of nodes affected by successful plan runs during the reporting period.</td>
</tr>
<tr>
<td>minutesFreedByPlanRuns</td>
<td>Total number of minutes freed by automated plan runs. This calculation is based on the average minutes saved per plan run, as specified by the minutesFreedPerPlanRun query parameter.</td>
</tr>
<tr>
<td>totalMinutesFreed</td>
<td>Total number of minutes free by all automated changes.</td>
</tr>
<tr>
<td>totalDollarsSaved</td>
<td>If the query specified an averageHourlySalary, total cost savings for all automated changes.</td>
</tr>
</tbody>
</table>

Examples

To generate a report for specified dates using the default time freed values:

```
curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: <RBAC_TOKEN>" \
-G "https://<HOSTNAME>/api/reports/value" \
--data-urlencode 'startDate=2020-07-08' \
--data-urlencode 'endDate=2020-07-15'
```

Result:

```
{
  "startDate": "2020-07-08",
  "endDate": "2020-07-15",
  "totalCorrectiveChanges": 0,
  "minutesFreedByCorrectiveChanges": 0,
  "totalIntentionalChanges": 18,
  "minutesFreedByIntentionalChanges": 1620,
}
To generate cost savings using default report dates and time freed values:

curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: <rbac token>" \
-G "https://<pe-console-fqdn>/api/reports/value" \
--data-urlencode 'averageHourlySalary=40'

Result:

```
{
    "startDate": "2020-07-08",
    "endDate": "2020-07-15",
    "totalCorrectiveChanges": 0,
    "minutesFreedByCorrectiveChanges": 0,
    "totalIntentionalChanges": 18,
    "minutesFreedByIntentionalChanges": 1620,
    "totalNodesAffectedByPlanRuns": 0,
    "totalNodesAffectedByTaskRuns": 0,
    "minutesFreedByPlan Runs": 0,
    "minutesFreedByTaskRuns": 0,
    "totalMinutesFreed": 1620,
    "totalDollarsSaved": 1080,
}
```

To generate a report with custom values for time freed:

curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: $(cat ~/.puppetlabs/token)" \
-G "https://<pe-console-fqdn>/api/reports/value" \
--data-urlencode 'minutesFreedPerCorrectiveChange=10' \
--data-urlencode 'minutesFreedPerIntentionalChange=20' \
--data-urlencode 'minutesFreedPerTaskRun=30' \
--data-urlencode 'minutesFreedPerPlanRun=40'

Result:

```
{
    "startDate": "2020-07-01",
    "endDate": "2020-07-08",
    "totalCorrectiveChanges": 1,
    "minutesFreedByCorrectiveChanges": 10,
    "totalIntentionalChanges": 2,
    "minutesFreedByIntentionalChanges": 40,
    "totalNodesAffectedByTaskRuns": 3,
    "minutesFreedByTaskRuns": 90,
    "totalNodesAffectedByPlanRuns": 4,
    "minutesFreedByPlan Runs": 160,
    "totalMinutesFreed": 300
}
```
Infrastructure reports

Each time Puppet runs on a node, it generates a report that provides information such as when the run took place, any issues encountered during the run, and the activity of resources on the node. These reports are collected on the Reports page in the console.

Working with the reports table

The Reports page provides a summary view of key data from each report. Use this page to track recent node activity so you can audit your system and perform root cause analysis over time.

The reports table lists the number of resources on each node in each of the following states:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correction applied</td>
<td>Number of resources that received a corrective change after Puppet identified resources that were out of sync with the applied catalog.</td>
</tr>
<tr>
<td>Failed</td>
<td>Number of resources that failed.</td>
</tr>
<tr>
<td>Changed</td>
<td>Number of resources that changed.</td>
</tr>
<tr>
<td>Unchanged</td>
<td>Number of resources that remained unchanged.</td>
</tr>
<tr>
<td>No-op</td>
<td>Number of resources that would have been changed if not run in no-op mode.</td>
</tr>
<tr>
<td>Skipped</td>
<td>Number of resources that were skipped because they depended on resources that failed.</td>
</tr>
<tr>
<td>Failed restarts</td>
<td>Number of resources that were supposed to restart but didn’t.</td>
</tr>
</tbody>
</table>

For example, if changes to one resource notify another resource to restart, and that resource doesn’t restart, a failed restart is reported. It’s an indirect failure that occurred in a resource that was otherwise unchanged.

The reports table also offers the following information:

- **No-op mode**: An indicator of whether the node was run in no-op mode.
- **Config retrieval**: Time spent retrieving the catalog for the node (in seconds).
- **Run time**: Time spent applying the catalog on the node (in seconds).

**Tip**: Report count caching is used to improve console performance. In some cases, caching might cause summary counts of available reports to be displayed inaccurately the first time the page is accessed after a fresh install.

Filtering reports

You can filter the list of reports displayed on the Reports page by run status and by node fact. If you set a run status filter, and also set a node fact filter, the table takes both filters into account, and shows only those reports matching both filters.

Clicking **Remove filter** removes all filters currently in effect.

The filters you set are persistent. If you set run status or fact filters on the Reports page, they continue to be applied to the table until they're changed or removed, even if you navigate to other pages in the console or log out. The persistent storage is associated with the browser tab, not your user account, and is cleared when you close the tab.

© 2022 Puppet, Inc., a Perforce company
Filter by node run status
Filter reports to quickly focus on nodes with failures or change events by using the **Filter by run status** bar.

1. Select a run status (such as **No-op mode: with failures**). The table updates to reflect your filter selection.
2. To remove the run status filter, select **All run statuses**.

Filter by node fact
You can create a highly specific list of nodes for further investigation by using the fact filter tool.

For example, you can check that nodes you’ve updated have successfully changed, or find out the operating systems or IP addresses of a set of failed nodes to better understand the failure. You might also filter by facts to fulfill an auditor’s request for information, such as the number of nodes running a particular version of software.

1. Click **Filter by fact value**. In the **Fact** field, select one of the available facts. An empty fact field is not allowed.

   **Tip:** To see the facts and values reported by a node on its most recent run, click the node name in the **Run status** table, then select the node’s **Facts** tab.

2. Select an Operator:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>is</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>is not</td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>matches a regular expression (regex)</td>
<td>Select this operator to use wildcards and other regular expressions if you want to find matching facts without having to specify the exact value.</td>
</tr>
<tr>
<td>!~</td>
<td>does not match a regular expression (regex)</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>Can be used only with facts that have a numeric value.</td>
</tr>
</tbody>
</table>

3. In the **Value** field, enter a value. Strings are case-sensitive, so make sure you use the correct case.

   The filter displays an error if you use an invalid string operator (for example, selecting a numeric value operator such as >= and entering a non-numeric string such as pilsen as the value) or enter an invalid regular expression.

   **Note:** If you enter an invalid or empty value in the **Value** field, PE takes the following action in order to avoid a filter error:
   - Invalid or empty Boolean facts are processed as **false**, and results are retrieved accordingly.
   - Invalid or empty numeric facts are processed as **0**, and results are retrieved accordingly.
   - Invalid or incomplete regular expressions invalidate the filter, and no results are retrieved.

4. Click **Add**.
5. As needed, repeat these steps to add additional filters. If filtering by more than one node fact, specify either **Nodes must match all rules** or **Nodes can match any rule**.

© 2022 Puppet, Inc., a Perforce company
Working with individual reports

To examine a report in greater detail, click Report time. This opens a page that provides details for the node’s resources in three sections: Events, Log, and Metrics.

Events

The Events tab lists the events for each managed resource on the node, its status, whether correction was applied to the resource, and — if it changed — what it changed from and what it changed to. For example, a user or a file might change from absent to present.

To filter resources by event type, click Filter by event status and choose an event.

Sort resources by name or events by severity level, ascending or descending, by clicking the Resource or Events sorting controls.

To download the events data as a .csv file, click Export data. The filename is events-<node name>-<timestamp>.

Log

The Log tab lists errors, warnings, and notifications from the node’s latest Puppet run.

Each message is assigned one of the following severity levels:

<table>
<thead>
<tr>
<th>Standard</th>
<th>Caution (yellow)</th>
<th>Warning (red)</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>warning</td>
<td>err</td>
</tr>
<tr>
<td>info</td>
<td>alert</td>
<td>emerg</td>
</tr>
<tr>
<td>notice</td>
<td></td>
<td>crit</td>
</tr>
</tbody>
</table>

To read the report chronologically, click the time sorting controls. To read it in order of issue severity, click the severity level sorting controls.

To download the log data as a .csv file, click Export data. The filename is log-<node name>-<timestamp>.

Metrics

The Metrics tab provides a summary of the key data from the node’s latest Puppet run.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report submitted by:</td>
<td>The certname of the primary server that submitted the report to PuppetDB.</td>
</tr>
<tr>
<td>Puppet environment</td>
<td>The environment assigned to the node.</td>
</tr>
</tbody>
</table>
## Analyzing changes across Puppet runs

The Events page in the console shows a summary of activity in your infrastructure. You can analyze the details of important changes, and investigate common causes behind related events. You can also examine specific class, node, and resource events, and find out what caused them to fail, change, or run as no-op.

### What is an event?

An event occurs whenever PE attempts to modify an individual property of a given resource. Reviewing events lets you see detailed information about what has changed on your system, or what isn't working properly.

During a Puppet run, Puppet compares the current state of each property on each resource to the desired state for that property, as defined by the node's catalog. If Puppet successfully compares the states and the property is already in sync (in other words, if the current state is the desired state), Puppet moves on to the next resource without noting anything. Otherwise, it attempts some action and records an event, which appears in the report it sends to the primary server at the end of the run. These reports provide the data presented on the Events page in the console.

---

### Metric Description

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
</table>
| Puppet run           | • The time that the Puppet run began  
                       | • The time that the primary server submitted the catalog  
                       | • The time that the Puppet run finished  
                       | • The time PuppetDB received the report  
                       | • The duration of the Puppet run  
                       | • The length of time to retrieve the catalog  
                       | • The length of time to apply the resources to the catalog |
| Catalog application  | Information about the catalog application that produces the report; the config version that Puppet uses to match a specific catalog for a node to a specific Puppet run, the catalog UUID that identifies the catalog used to generate a report during a Puppet run, and whether the Puppet run used a cached catalog. |
| Resources            | The total number of resources in the catalog.                               |
| Events               | A list of event types and the total count for each one.                     |
| Top resource types   | A list of the top resource types by time, in seconds, it took to be applied. |
Event types

There are six types of event that can occur when Puppet reviews each property in your system and attempts to make any needed changes. If a property is already in sync with its catalog, no event is recorded: no news is good news in the world of events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure</td>
<td>A property was out of sync; Puppet tried to make changes, but was unsuccessful.</td>
</tr>
<tr>
<td>Corrective change</td>
<td>Puppet found an inconsistency between the last applied catalog and a property's configuration, and corrected the property to match the catalog.</td>
</tr>
<tr>
<td>Intentional change</td>
<td>Puppet applied catalog changes to a property.</td>
</tr>
<tr>
<td>Corrective no-op</td>
<td>Puppet found an inconsistency between the last applied catalog and a property's configuration, but Puppet was instructed to not make changes on this resource, via either the <code>--noop</code> command-line option, the <code>noop</code> setting, or the <code>noop =&gt; true</code> metaparameter. Instead of making a corrective change, Puppet logs a corrective no-op event and reports the change it would have made.</td>
</tr>
<tr>
<td>Intentional no-op</td>
<td>Puppet would have applied catalog changes to a property, but Puppet was instructed to not make changes on this resource, via either the <code>--noop</code> command-line option, the <code>noop</code> setting, or the <code>noop =&gt; true</code> metaparameter. Instead of making an intentional change, Puppet logs an intentional no-op event and reports the change it would have made.</td>
</tr>
<tr>
<td>Skip</td>
<td>A prerequisite for this resource was not met, so Puppet did not compare its current state to the desired state. This prerequisite is either one of the resource's dependencies or a timing limitation set with the <code>schedule</code> metaparameter. The resource might be in sync or out of sync; Puppet doesn't know yet. If the <code>schedule</code> metaparameter is set for a given resource, and the scheduled time hasn't arrived when the run happens, that resource logs a skip event on the Events page. This is true for a user-defined <code>schedule</code>, but does not apply to built-in scheduled tasks that happen weekly, daily, or at other intervals.</td>
</tr>
</tbody>
</table>

Working with the Events page

During times when your deployment is in a state of stability, with no changes being made and everything functioning optimally, the Events page reports little activity, and might not seem terribly interesting. But when change occurs—when packages require upgrades, when security concerns threaten, or when systems fail—the Events page helps you understand what's happening and where so you can react quickly.

The Events page fetches data when loading, and does not refresh—even if there's a Puppet run while you're on the page—until you close or reload the page. This ensures that shifting data won't disrupt an investigation.

You can see how recent the shown data is by checking the timestamp at the top of the page. Reload the page to update the data to the most recent events.
Tip: Keeping time synchronized by running NTP across your deployment helps the Events page produce accurate information. NTP is easily managed with PE, and setting it up is an excellent way to learn Puppet workflows.

Monitoring infrastructure with the Events summary pane

The Events page displays all events from the latest report of every responsive node in the deployment.

Tip: By default, PE considers a node unresponsive after one hour, but you can configure this setting to meet your needs by adjusting the puppet_enterprise::console_services::no_longer_reporting_cutoff parameter.

On the left side of the screen, the Events summary pane shows an overview of Puppet activity across your infrastructure. This data can help you rapidly assess the magnitude of any issue.

The Events summary pane is split into three categories—the Classes summary, Nodes summary, and Resources summary—to help you investigate how a change or failure event impacts your entire deployment.

Gaining insight with the Events detail pane

Clicking an item in the Events summary pane loads its details (and any sub-items) in the Events detail pane on the right of the screen. The summary pane on the left always shows the list of items from which the one in the detail pane on the right was chosen, to let you easily view similar items and compare their states.

Click any item in the the Classes summary, Nodes summary, or Resources summary to load more specific info into the detail pane and begin looking for the causes of notable events. Switch between perspectives to find the common threads among a group of failures or corrective changes, and follow them to a root cause.

Analyzing changes and failures

You can use the Events page to analyze the root causes of events resulting from a Puppet run. For example, to understand the cause of a failure after a Puppet run, select the class, node, or resource with a failure in the Events summary pane, and then review the details of the failure in the Events detail pane.

You can view additional details by clicking on the failed item in the in the Events detail pane.

Use the Classes summary, Nodes summary, and Resources summary to focus on the information you need. For example, if you’re concerned about a failed service, say Apache or MongoDB, you can start by looking into failed resources or classes. If you’re experiencing a geographic outage, you might start by drilling into failed node events.

Understanding event display issues

In some special cases, events are not displayed as expected on the Events page. These cases are often caused by the way that the console receives data from other parts of Puppet Enterprise, but sometimes are due to the way your Puppet code is interpreted.

Runs that restart PuppetDB are not displayed

If a given Puppet run restarts PuppetDB, Puppet is not able to submit a run report from that run to PuppetDB because PuppetDB is not available. Because the Events page relies on data from PuppetDB, and PuppetDB reports are not queued, the Events page does not display any events from that run. Note that in such cases, a run report is available on the Reports page. Having a Puppet run restart PuppetDB is an unlikely scenario, but one that could arise in cases where some change to, say, a parameter in the puppet::db class causes the pe::puppet::db service to restart.

Runs without a compiled catalog are not displayed

If a run encounters a catastrophic failure where an error prevents a catalog from compiling, the Events page does not display any failures. This is because no events occurred.

© 2022 Puppet, Inc., a Perforce company
Simplified display for some resource types

For resource types that take the ensure property, such as user or file resource types, the Events page displays a single event when the resource is first created. This is because Puppet has changed only one property (ensure), which sets all the baseline properties of that resource at the same time. For example, all of the properties of a given user are created when the user is added, just as if the user was added manually. If a later Puppet run changes properties of that user resource, each individual property change is shown as a separate event.

Updated modes display without leading zeros

When the mode attribute for a file resource is updated, and numeric notation is used, leading zeros are omitted in the New Value field on the Events page. For example, 0660 is shown as 660 and 0000 is shown as 0.

Puppet Enterprise metrics

You can use Puppet Enterprise (PE) metrics for your own performance tuning or provide them to Support for troubleshooting.

Related information

Viewing and managing Puppet Server metrics on page 415

Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

About the puppet_metrics_collector module

The puppet_metrics_collector module can collect metrics from the status endpoints of Puppet Enterprise (PE) services.

The puppet_metrics_collector module is installed with PE and is disabled by default.

Important: If you have a version of this module, from the Forge or other sources, specified in the code, you must remove this version before upgrading to allow the version bundled with PE to be asserted.

Two parameters control metrics collection:

- puppet_enterprise::enable_metrics_collection: Collect metrics from PE services, such as Puppet Server and PuppetDB.
- puppet_enterprise::enable_system_metrics_collection: Collect metrics from the operating system the PE services run on.

Visit the puppet_metrics_collector Forge page to learn about this module's other classes and features, such as retention time, collection frequency, and parameters for specific services.

You can use the puppet_operational_dashboards module to view PE metrics.

Enable or disable metrics collection

The puppet_metrics_collector module is disabled by default, and you can manually configure the metrics collection parameters.

These parameters accept Boolean values.

1. Set these parameters to true to enable each category of metrics collection:

   - puppet_enterprise::enable_metrics_collection: Control metrics collection for Puppet Enterprise (PE) services.
   - puppet_enterprise::enable_system_metrics_collection: Control metrics collection for the operating system the PE services run on.

2. Run Puppet: puppet agent -t

Metrics collection is now enabled. To disable one or both streams of metrics collection, set either or both parameters to false and run Puppet.
Related information
Methods for configuring PE on page 286
After you've installed Puppet Enterprise (PE), optimize it for your environment by configuring and tuning settings as needed. For example, you might want to add your certificate to the allowlist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

Viewing and managing Puppet Server metrics

Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

You can retrieve, track, and visualize Puppet Server metrics with:

- The Puppet Report Viewer app for Splunk. You'll need the splunk_hec and pe_event_forwarding modules along with the app. For information about this option, refer to these blog posts:
  - PE Metrics in Splunk: Puppet Report Viewer 3.1
  - Introducing Puppet and Splunk integrations to improve reporting speed and scale

- The puppet_operational_dashboards module (which you can use along with the puppet_metrics_collector module that is already bundle with PE).

- Customizable, networked Graphite and Grafana instances. While the grafanadash and puppet-graphite modules are not Puppet-supported modules (they are provided for testing and demonstration purposes only), you can learn about these options in Getting started with Graphite on page 415 and Available Graphite metrics on page 419.

- The Status API on page 436 and Metrics API on page 449 endpoints.

- Getting started with Graphite on page 415
Puppet Enterprise can export many metrics to Graphite, a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. After Graphite support is enabled, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

- Available Graphite metrics on page 419
These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

Getting started with Graphite

Puppet Enterprise can export many metrics to Graphite, a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. After Graphite support is enabled, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

Note: A Graphite setup is deeply customizable and can report many different Puppet Server metrics on demand. However, it requires considerable configuration and additional server resources. For an easier, but more limited, web-based dashboard of Puppet Server metrics built into Puppet Server, use the developer dashboard. To retrieve metrics manually via HTTP, use the Status API.

To use Graphite with Puppet Enterprise, you must:

- Install and configure a Graphite server.
- Enable Puppet Server's Graphite support

Grafana provides a web-based customizable dashboard that's compatible with Graphite, and the Grafanadash module installs and configures it by default.
Using the Grafanadash module
The Grafanadash module quickly installs and configures a basic test instance of Graphite with the Grafana extension. When installed on a dedicated Puppet agent, this module provides a quick demonstration of how Graphite and Grafana can consume and display Puppet Server metrics.

**CAUTION:** The Grafanadash module referenced in this document is not a Puppet-supported module; it is for testing and demonstration purposes only. It is tested against CentOS 7 only. Also, install this module on a dedicated agent only. Do not install it on the primary server, because the module makes security policy changes that are inappropriate for a primary server:

- SELinux can cause issues with Graphite and Grafana, so the module temporarily disables SELinux. If you reboot the machine after using the module to install Graphite, you must disable SELinux again and restart the Apache service to use Graphite and Grafana.
- The module disables the iptables firewall and enables cross-origin resource sharing on Apache, which are potential security risks.

Installing the Grafanadash module
Install the Grafanadash module on a *nix agent. The module's grafanadash::dev class installs and configures a Graphite server, the Grafana extension, and a default dashboard.

1. Install a *nix PE agent to serve as the Graphite server.
2. On the Puppet agent node, run: `sudo puppet module install puppetlabs-grafanadash`
3. On the Puppet agent node, run: `sudo puppet apply -e 'include grafanadash::dev'`

Running Grafana
Grafana is a dashboard that can interpret and visualize Puppet Server metrics over time, but you must configure it to do so.

Grafana runs as a web dashboard, and the Grafanadash module configures it at port 10000 by default. However, there are no Puppet metrics displayed by default. You must create a metrics dashboard to view Puppet's metrics in Grafana, or edit and import a JSON-based dashboard such as the sample Grafana dashboard that we provide.

1. In a web browser on a computer that can reach the Puppet agent node, navigate to `http://<AGENT_HOSTNAME>:10000`.
2. Open the `sample_metrics_dashboard.json` file in a text editor on the same computer you're using to access Grafana.
3. Throughout the file, replace our sample setting of `primary.example.com` with the hostname of your primary server. This value must be used as the `metrics_server_id` setting, as configured below.
4. Save the file.
5. In the Grafana UI, click search (the folder icon), then Import, then Browse.
6. Navigate to and select the edited JSON file.

This loads a dashboard with nine graphs that display various metrics exported from the Puppet Server to the Graphite server. However, these graphs remain empty until you enable Puppet Server's Graphite metrics.

**Related information**
Sample Grafana dashboard graphs on page 417
Use the sample Grafana dashboard as your starting point and customize it to suit your needs. You can click on the title of any graph, and then click edit to adjust the graphs as you see fit.

Enabling Puppet Server's Graphite support
Use the PE Master node group in the console to configure Puppet Server's metrics output settings.

1. In the console, click Node groups, and in the PE Infrastructure group, select the PE Master group.
2. On the **Classes** tab, in the `puppet_enterprise::profile::master` class, add these parameters:
   a) Set `metrics_graphite_enabled` to `true` (default is `false`).
   b) Set `metrics_server_id` to the primary server hostname.
   c) Set `metrics_graphite_host` to the hostname for the agent node on which you're running Graphite and Grafana.
   d) Set `metrics_graphite_update_interval_seconds` to a value to set Graphite's update frequency in seconds. This setting is optional, and the default value is `60`.

3. Verify that these parameters are set to their default values, unless your Graphite server uses a non-standard port:
   a) Set `metrics_jmx_enabled` to `true` (default value).
   b) Set `metrics_graphite_port` to `2003` (default value) or the Graphite port on your Graphite server.
   c) Set `profiler_enabled` to `true` (default value).

4. Commit changes.

**Sample Grafana dashboard graphs**

Use the sample Grafana dashboard as your starting point and customize it to suit your needs. You can click on the title of any graph, and then click **edit** to adjust the graphs as you see fit.

**Sample Grafana dashboard code**

<table>
<thead>
<tr>
<th>Graph name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active requests</td>
<td>This graph serves as a &quot;health check&quot; for the Puppet Server. It shows a flat line that represents the number of CPUs you have in your system, a metric that indicates the total number of HTTP requests actively being processed by the server at any moment in time, and a rolling average of the number of active requests. If the number of requests being processed exceeds the number of CPUs for any significant length of time, your server might be receiving more requests than it can efficiently process.</td>
</tr>
<tr>
<td>Request durations</td>
<td>This graph breaks down the average response times for different types of requests made by Puppet agents. This indicates how expensive catalog and report requests are compared to the other types of requests. It also provides a way to see changes in catalog compilation times when you modify your Puppet code. A sharp curve upward for all of the types of requests indicates an overloaded server, and they should trend downward after reducing the load on the server.</td>
</tr>
<tr>
<td>Request ratios</td>
<td>This graph shows how many requests of each type that Puppet Server has handled. Under normal circumstances, you should see about the same number of catalog, node, or report requests, because these all happen one time per agent run. The number of file and file metadata requests correlate to how many remote file resources are in the agents' catalogs.</td>
</tr>
<tr>
<td>External HTTP Communications</td>
<td>This graph tracks the amount of time it takes Puppet Server to send data and requests for common operations to, and receive responses from, external HTTP services, such as PuppetDB.</td>
</tr>
<tr>
<td>File Sync</td>
<td>This graph tracks how long Puppet Server spends on File Sync operations, for both its storage and client services.</td>
</tr>
<tr>
<td>Graph name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>JRubies</td>
<td>This graph tracks how many JRubies are in use, how many are free, the mean number of free JRubies, and the mean number of requested JRubies. If the number of free JRubies is often less than one, or the mean number of free JRubies is less than one, Puppet Server is requesting and consuming more JRubies than are available. This overload reduces Puppet Server's performance. While this might simply be a symptom of an under-resourced server, it can also be caused by poorly optimized Puppet code or bottlenecks in the server's communications with PuppetDB if it is in use. If catalog compilation times have increased but PuppetDB performance remains the same, examine your Puppet code for potentially unoptimized code. If PuppetDB communication times have increased, tune PuppetDB for better performance or allocate more resources to it. If neither catalog compilation nor PuppetDB communication times are degraded, the Puppet Server process might be under-resourced on your server. If you have available CPU time and memory, increase the number of JRuby instances to allow it to allocate more JRubies. Otherwise, consider adding additional compilers to distribute the catalog compilation load.</td>
</tr>
</tbody>
</table>
| JRuby Timers | This graph tracks several JRuby pool metrics.  
• The borrow time represents the mean amount of time that Puppet Server uses ("borrows") each JRuby from the pool.  
• The wait time represents the total amount of time that Puppet Server waits for a free JRuby instance.  
• The lock held time represents the amount of time that Puppet Server holds a lock on the pool, during which JRubies cannot be borrowed. This occurs while Puppet Server synchronizes code for File Sync.  
• The lock wait time represents the amount of time that Puppet Server waits to acquire a lock on the pool.  
These metrics help identify sources of potential JRuby allocation bottlenecks. |
| Memory Usage | This graph tracks how much heap and non-heap memory that Puppet Server uses. |
### Example Grafana dashboard excerpt

The following example shows only the `targets` parameter of a dashboard to demonstrate the full names of Puppet's exported Graphite metrics (assuming the Puppet Server instance has a domain of `primary.example.com`) and a way to add targets directly to an exported Grafana dashboard's JSON content.

```json
"panels": [
  {
    "span": 4,
    "editable": true,
    "type": "graphite",
    ...
    "targets": [
      {
        "target": "alias(puppetlabs.primary.example.com.num-cpus,'num cpus')"
      },
      {
        "target": "alias(puppetlabs.primary.example.com.http.active-requests.count,'active requests')"
      },
      {
        "target": "alias(puppetlabs.primary.example.com.http.active-histo.mean,'average')"
      },
      "aliasColors": {},
      "aliasYAxis": {},
      "title": "Active Requests"
    }
  }
]
```

See the sample Grafana dashboard for a detailed example of how a Grafana dashboard accesses these exported Graphite metrics.

### Available Graphite metrics

These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

### Graphite metrics properties

Each metric is prefixed with `puppetlabs.<PRIMARY_HOSTNAME>`; for instance, the Grafana dashboard file refers to the `num-cpus` metric as `puppetlabs.<PRIMARY_HOSTNAME>.num-cpus`.

Additionally, metrics might be suffixed by fields, such as `count` or `mean`, that return more specific data points. For instance, the `puppetlabs.<PRIMARY_HOSTNAME>.compiler.mean` metric returns only the mean length of time it takes Puppet Server to compile a catalog.

To aid with reference, metrics in the list below are segmented into three groups:
• **Statistical metrics:** Metrics that have all eight of these statistical analysis fields, in addition to the top-level metric:
  
  • **max:** Its maximum measured value.
  • **min:** Its minimum measured value.
  • **mean:** Its mean, or average, value.
  • **stddev:** Its standard deviation from the mean.
  • **count:** An incremental counter.
  • **p50:** The value of its 50th percentile, or median.
  • **p75:** The value of its 75th percentile.
  • **p95:** The value of its 95th percentile.

• **Counters only:** Metrics that only count a value, or only have a `count` field.

• **Other:** Metrics that have unique sets of available fields.

**Note:**

Puppet Server can export many metrics—so many that past versions of Puppet Enterprise could overwhelm Grafana servers. As of Puppet Enterprise 2016.4, Puppet Server exports only a subset of its available metrics by default. This set is designed to report the most relevant Puppet Server metrics for administrators monitoring its performance and stability.

To add to the default list of exported metrics, see Modifying Puppet Server’s exported metrics.

Puppet Server exports each metric in the lists below by default.

**Statistical metrics**

Compiler metrics:

• `puppetlabs.<PRIMARY_HOSTNAME>.compiler`: The time spent compiling catalogs. This metric represents the sum of the `compiler.compile`, `static_compile`, `find_facts`, and `find_node` fields.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.compile`: The total time spent compiling dynamic (non-static) catalogs.

  To measure specific nodes and environments, see Modifying Puppet Server’s exported metrics.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.find_facts`: The time spent parsing facts.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.find_node`: The time spent retrieving node data. If the Node Classifier (or another ENC) is configured, this includes the time spent communicating with it.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile`: The time spent compiling static catalogs.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile_inlining`: The time spent inlining metadata for static catalogs.

  • `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile_postprocessing`: The time spent post-processing static catalogs.

File sync metrics:

• `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.clone-timer`: The time spent by file sync clients on compilers initially cloning repositories on the primary server.

• `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.fetch-timer`: The time spent by file sync clients on compilers fetching repository updates from the primary server.

• `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.sync-clean-check-timer`: The time spent by file sync clients on compilers checking whether the repositories are clean.

• `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.sync-timer`: The time spent by file sync clients on compilers synchronizing code from the private datadir to the live codedir.

© 2022 Puppet, Inc., a Perforce company
• puppetlabs.<PRIMARY_HOSTNAME>.file-sync-storage.commit-timer: The time spent committing code on the primary server into the file sync repository.

Function metrics:
• puppetlabs.<PRIMARY_HOSTNAME>.functions: The amount of time during catalog compilation spent in function calls. The functions metric can also report any of the statistical metrics fields for a single function by specifying the function name as a field.

  For example, to report the mean time spent in a function call during catalog compilation, use puppetlabs.<PRIMARY_HOSTNAME>.functions.<FUNCTION-NAME>.mean.

HTTP metrics:
• puppetlabs.<PRIMARY_HOSTNAME>.http.active-histo: A histogram of active HTTP requests over time.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-catalog-/*/-requests: The time Puppet Server has spent handling catalog requests, including time spent waiting for an available JRuby instance.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment-/*/-requests: The time Puppet Server has spent handling environment requests, including time spent waiting for an available JRuby instance.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment_classes-/*/-requests: The time spent handling requests to the environment_classes API endpoint, which the Node Classifier uses to refresh classes.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environments-requests: The time spent handling requests to the environments API endpoint requests made by the Orchestrator.

  The following metrics measure the time spent handling file-related API endpoints:
  • puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-requests
  • puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_content-/*/-requests
  • puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadata-/*/-requests
  • puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadatas-/*/-requests
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-node-/*/-requests: The time spent handling node requests, which are sent to the Node Classifier. A bottleneck here might indicate an issue with the Node Classifier or PuppetDB.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-report-/*/-requests: The time spent handling report requests. A bottleneck here might indicate an issue with PuppetDB.
• puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-static_file_content-/*/-requests: The time spent handling requests to the static_file_content API endpoint used by Direct Puppet with file sync.

JRuby metrics: Puppet Server uses an embedded JRuby interpreter to execute Ruby code. JRuby spawns parallel instances known as JRubies to execute Ruby code, which occurs during most Puppet Server activities. See Tuning JRuby on Puppet Server for details on adjusting JRuby settings.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-timer: The time spent with a borrowed JRuby.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.free-jrubies-histo: A histogram of free JRubies over time. This metric's average value must be greater than 1; if it isn't, more JRubies or another compiler might be needed to keep up with requests.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.lock-held-timer: The time spent holding the JRuby lock.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.lock-wait-timer: The time spent waiting to acquire the JRuby lock.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.requested-jrubies-histo: A histogram of requested JRubies over time. This increases as the number of free JRubies, or the free-jrubies-histo metric, decreases, which can suggest that the server's capacity is being depleted.
• puppetlabs.<PRIMARY_HOSTNAME>.jruby.wait-timer: The time spent waiting to borrow a JRuby.
PuppetDB metrics: The following metrics measure the time that Puppet Server spends sending or receiving data from PuppetDB.

- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.catalog.save
- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.command.submit
- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.facts.find
- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.facts.search
- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.report.process
- puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.resource.search

Counters only

HTTP metrics:

- puppetlabs.<PRIMARY_HOSTNAME>.http.active-requests: The number of active HTTP requests.
- The following counter metrics report the percentage of each HTTP API endpoint's share of total handled HTTP requests.
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-catalog-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment_classes-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environments-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_content-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadata-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadatas-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-node-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-report-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-resource_type-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-resource_types-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-static_file_content-/*/-percentage
  - puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-status-/*/-percentage
- puppetlabs.<PRIMARY_HOSTNAME>.http.total-requests: The total requests handled by Puppet Server.

JRuby metrics:

- puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-count: The number of successfully borrowed JRubies.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-retry-count: The number of attempts to borrow a JRuby that must be retried.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-timeout-count: The number of attempts to borrow a JRuby that resulted in a timeout.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.request-count: The number of requested JRubies.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.return-count: The number of JRubies successfully returned to the pool.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.num-free-jrubies: The number of free JRuby instances. If this number is often 0, more requests are coming in than the server has available JRuby instances. To alleviate this, increase the number of JRuby instances on the Server or add additional compilers.
- puppetlabs.<PRIMARY_HOSTNAME>.jruby.num-jrubies: The total number of JRuby instances on the server, governed by the max-active-instances setting. See Tuning JRuby on Puppet Server for details.
Other metrics

These metrics measure raw resource availability and capacity.

- puppetlabs.<PRIMARY_HOSTNAME>.num-cpus: The number of available CPUs on the server.
- puppetlabs.<PRIMARY_HOSTNAME>.uptime: The Puppet Server process's uptime.
- Total, heap, and non-heap memory that's committed (committed), initialized (init), and used (used), and the maximum amount of memory that can be used (max).
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.total.committed
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.total.init
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.total.used
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.total.max
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.committed
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.init
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.used
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.max
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.committed
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.init
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.used
  - puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.max

Related information

Modifying exported metrics on page 423

In addition to the default metrics, you can also export metrics measuring specific environments and nodes managed by Puppet Server.

Modifying exported metrics

In addition to the default metrics, you can also export metrics measuring specific environments and nodes managed by Puppet Server.

The $metrics_puppetserver_metrics_allowed class parameter in the puppet_enterprise::profile::master class takes an array of metrics as strings. To export additional metrics, add them to this array.

Optional metrics include:

- compiler.compile.<ENVIRONMENT> and compiler.compile.<ENVIRONMENT>.<NODE-NAME>, and all statistical fields suffixed to these (such as compiler.compile.<ENVIRONMENT>.mean).
- compiler.compile.evaluate_resources.<RESOURCE>: Time spent evaluating a specific resource during catalog compilation.

Omit the puppetlabs.<MASTER-HOSTNAME> prefix and field suffixes (such as .count or .mean) from metrics. Instead, suffix the environment or node name as a field to the metric.

1. For example, to track the compilation time for the production environment, add compiler.compile.production to the metrics-allowed list.
2. To track only the my.node.localdomain node in the production environment, add compiler.compile.production.my.node.localdomain to the metrics-allowed list.

Sample Grafana dashboard JSON code

The following JSON code is an example metrics dashboard. Copy and save it to a .json file to reuse it.

For details, see Sample Grafana dashboard graphs on page 417

```json
{
    "title": "master",
    "services": {
```
"nullPointMode": "connected",
"steppedLine": false,
"tooltip": {
  "value_type": "cumulative",
  "query_as_alias": true
},
"targets": [
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-catalog-.*.-requests.mean,'catalog')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-report-.*.-requests.mean,'report')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-node-.*.-requests.mean,'node')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-file_metadatas-.*.-requests.mean,'file_metadatas')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-file_metadata-.*.-requests.mean,'file_metadata')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.puppet-v3-file_content-.*.-requests.mean,'file_content')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.other-requests.mean,'other')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.http.total-requests.mean,'all')"
  }
],
"aliasColors": {},
"aliasYAxis": {},
"title": "Request Durations"
},
{ "span": 4,
  "editable": true,
  "type": "graphite",
  "loadingEditor": false,
  "datasource": null,
  "renderer": "flot",
  "x-axis": true,
  "y-axis": true,
  "scale": 1,
  "y_formats": [
    "short",
    "short"
  ],
  "grid": {
    "max": null,
    "min": null,
    "threshold1": null,
    "threshold2": null,
    "threshold1Color": "rgba(216, 200, 27, 0.27)"}
"threshold2Color": "rgba(234, 112, 112, 0.22)",
"thresholdLine": false
},
"annotate": {
  "enable": false
},
"resolution": 100,
"lines": true,
"fill": 0,
"linewidth": 1,
"points": false,
"pointradius": 5,
"bars": false,
"stack": false,
"legend": {
  "show": true,
  "values": false,
  "min": false,
  "max": false,
  "current": false,
  "total": false,
  "avg": false
},
"percentage": true,
"zerofill": true,
"nullPointMode": "connected",
"steppedLine": false,
"tooltip": {
  "value_type": "individual",
  "query_as_alias": true
},
"targets": [
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-catalog-.*/percentage,100),'catalog')",
    "hide": false
  },
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-report-.*/percentage,100),'report')"
  },
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-node-.*/percentage,100),'node')"
  },
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-file_metadatas-.*/percentage,100),'file_metadatas')"
  },
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-file_metadata-.*/percentage,100),'file_metadata')"
  },
  {
    "target":
    "alias(scale(puppetlabs.master.example.com.http.puppet-v3-file_content-.*/percentage,100),'file_content')"
  }
],
"query_as_alias": true,
"targets": [
  {
    "target":
    "alias(puppetlabs.master.example.com.compiler.compile.mean,'compile')"
  },
  {
    "target":
    "alias(puppetlabs.master.example.com.functions.mean,'function calls')"
  },
  {
    "target":
    "alias(puppetlabs.master.example.com.compiler.find_node.mean,'find_node')"
  },
  {
    "target":
    "alias(puppetlabs.master.example.com.compiler.static_compile.mean,'static compile')"
  },
  {
    "target":
    "alias(puppetlabs.master.example.com.compiler.static_compile_postprocessing.mean,'static compile postprocessing')"
  },
  {
    "target":
    "alias(puppetlabs.master.example.com.compiler.find_facts.mean,'find_facts')"
  }
],
"aliasColors": {
  "create scope": "#BADFF4"
},
"aliasYAxis": {},
"title": "Compilation"}
"linewidth": 1,
"points": false,
"pointradius": 5,
"bars": false,
"stack": false,
"legend": {
  "show": true,
  "values": false,
  "min": false,
  "max": false,
  "current": false,
  "total": false,
  "avg": false
},
"percentage": false,
"zerofill": true,
"nullPointMode": "connected",
"steppedLine": false,
"tooltip": {
  "value_type": "cumulative",
  "query_as_alias": true
},
"targets": [
  {
    "target": "alias(puppetlabs.master.example.com.puppetdb.catalog.save.mean,'catalog save')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.puppetdb.command.submit.mean,'command submit')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.puppetdb.report.process.mean,'report process')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.puppetdb.facts.find.mean,'facts find')"
  }
],
"aliasColors": {},
"aliasYAxis": {},
"title": "Communications with PuppetDB"}
Monitoring and reporting

```json
"collapsible": true,
"panels": [
{
"span": 4,
"editable": true,
"type": "graphite",
"loadingEditor": false,
"datasource": null,
"renderer": "flot",
"x-axis": true,
"y-axis": true,
"scale": 1,
"y_formats": [
  "short",
  "short"
],
"grid": {
  "max": null,
  "min": 0,
  "threshold1": null,
  "threshold2": null,
  "threshold1Color": "rgba(216, 200, 27, 0.27)",
  "threshold2Color": "rgba(234, 112, 112, 0.22)"
},
"annotate": {
  "enable": false
},
"resolution": 100,
"lines": true,
"fill": 0,
"linewidth": 1,
"points": false,
"pointradius": 5,
"bars": false,
"stack": false,
"legend": {
  "show": true,
  "values": false,
  "min": false,
  "max": false,
  "current": false,
  "total": false,
  "avg": false
},
"percentage": false,
"zerofill": true,
"nullPointMode": "connected",
"steppedLine": false,
"tooltip": {
  "value_type": "cumulative",
  "query_as_alias": true
},
"targets": [
  {"target": "alias(puppetlabs.master.example.com.jruby.num-jrubies,'num jrubies')"},
  {"target": "alias(puppetlabs.master.example.com.jruby.num-free-jrubies,'num free jrubies')"},
  {"target": "alias(puppetlabs.master.example.com.jruby.free-jrubies-histo.mean,'average free jrubies')"
```
"zeroFill": true,
"nullPointMode": "connected",
"steppedLine": false,
"tooltip": {
  "value_type": "cumulative",
  "query_as_alias": true
},
"targets": [
  {
    "target": "alias(puppetlabs.master.example.com.memory.heap.used,'heap used')"
  },
  {
    "target": "alias(puppetlabs.master.example.com.memory.non-heap.used,'non-heap used')"
  }
],
"aliasColors": {},
"aliasYAxis": {},
"title": "Memory Usage",
"leftYAxisLabel": "",
"notice": false
],
"editable": true,
"failover": false,
"panel_hints": true,
"style": "dark",
"pulldowns": [
  {
    "type": "filtering",
    "collapse": false,
    "notice": false,
    "enable": false
  },
  {
    "type": "annotations",
    "enable": false
  }
],
"nav": [
  {
    "type": "timepicker",
    "collapse": false,
    "notice": false,
    "enable": true,
    "status": "Stable",
    "time_options": [
      "5m",
      "15m",
      "1h",
      "6h",
      "12h",
      "24h",
      "2d",
      "7d",
      "30d"
    ],
    "refresh_intervals": [
      "5s",
      "10s",
      "30s",
      "5m",
      ...
Status API

The status API allows you to check the health of PE components and services. It can be useful for automated monitoring of your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

There are four API endpoints that each contain different services on their unique port. You can check the overall health of all services within a given port, or the details of an individual service on a port.

**Console-services status API on port 4433**

Returns status details about the activity service, the classifier, and RBAC.

**Puppet Server status API on port 8140**

Returns status details about Code Manager, file sync client, file sync storage, and Puppet Server.

**Orchestrator status API on port 8143**

Returns status details about the orchestrator, PCP broker, and PCP broker v2.

**PuppetDB status API on port 8081**

Returns status details about PuppetDB.

The endpoints provide overview health information in an overall healthy/error/unknown status field, and fine-detail information such as the availability of the database, the health of other required services, or connectivity to the primary server.
• **Authenticating to the status API** on page 437
  Token-based authentication is not required to access the status API. You can choose to authenticate requests by using certificates, or you can access the API without authentication via HTTP.

• **Forming requests to the status API** on page 437
  When forming requests to the status API, specify the port that corresponds to the service you want to view the status for.

• **Status API: services endpoint** on page 438
  The /services endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

• **Status API: services plaintext endpoint** on page 442
  The activity service plaintext endpoints are designed for load balancers that don't support any kind of JSON parsing or parameter setting. They return simple string bodies (either the state of the service in question or a simple error message) and a status code relevant to the status result.

• **Status API: metrics endpoint** on page 443
  Puppet Server is capable of tracking advanced metrics to give you additional insight into its performance and health.

  Metrics API on page 449
  Puppet Enterprise includes an optional, enabled-by-default web endpoint for Java Management Extension (JMX) metrics, namely managed beans (MBeans).

**Related information**

API index on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

---

**Authenticating to the status API**

Token-based authentication is not required to access the status API. You can choose to authenticate requests by using certificates, or you can access the API without authentication via HTTP.

You can authenticate requests using a certificate listed in RBAC's certificate allowlist, located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. The certificate allowlist is a simple, flat file consisting of certnames that match the host, for example:

```
node1.example
node2.example
node3.example
```

**Note:** If you edit the certificate allowlist, you must reload the `pe-console-services` service (run `sudo service pe-console-services reload`) for your changes to take effect.

The status API's endpoints can be served over HTTP, which does not require any authentication. This is disabled by default.

**Tip:** To use HTTP, locate the PE Console node group in the console, and in the `puppet_enterprise::profile::console` class, set `console_services_plaintext_status_enabled` to `true`.

---

**Forming requests to the status API**

When forming requests to the status API, specify the port that corresponds to the service you want to view the status for.

The path prefix is /status, so, for example, the URL to get the statuses for all services as JSON on port 8140 is `https://<DNS NAME OF YOUR CONSOLE HOST>:8140/status/v1/services`. This retrieves the statuses for Code Manager and Puppet Server.

To access that URL using curl commands, run:

```
cert="$(puppet config print hostcert)"
```
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):8140/status/v1/services"
curl --cert "$cert" --cacert "$cacert" --key "$key" "$uri"

See Usage notes for curl examples for information about forming curl commands.

If enabled, the HTTP status endpoints are available on port 8123. See Authenticating to the status API on page 437 to enable HTTP.

Tip: To change the port, locate the PE Console node group in the console, and in the puppet_enterprise::profile::console class, set the console_services_plaintext_status_port parameter to your desired port number.

Status API: services endpoint

The /services endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

GET /status/v1/services
Use the /services endpoint to retrieve the statuses of PE services on a specified port.

The content type for this endpoint is application/json; charset=utf-8.

Request format

The request accepts the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>How thorough of a check to run. Set to critical, debug, or info (default).</td>
</tr>
<tr>
<td>timeout</td>
<td>Specified in seconds; defaults to 30.</td>
</tr>
</tbody>
</table>

The Status API endpoints return the status of services based on the port. Services available on each port are as follows:

<table>
<thead>
<tr>
<th>Port</th>
<th>Service(s)</th>
</tr>
</thead>
</table>
| 4433  | • RBAC  
|       | • Activity Service  
|       | • Classifier  |
| 8140  | • Code Manager  
|       | • file sync client  
|       | • file sync storage  
|       | • Puppet Server  |
For example, this request fetches the info status of all the console services running on port 4433, which are RBAC, the Activity Service, and the Classifier:

curl --insecure -H "X-Authentication: ALmWRLko6nJhi7bRQX4Qe-hOp07d4ZqDvHcfertiK81K8" \
--header "Content-Type: application/json" \
--request GET -G "https://puppet.status.example:4433/status/v1/services" \
--data-urlencode '{"level": "info", "timeout": "30"}'

Response format
The response is a JSON object that lists details about the services, using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_version</td>
<td>Package version of the JAR file containing a given service.</td>
</tr>
<tr>
<td>service_status_version</td>
<td>The version of the API used to report the status of the service.</td>
</tr>
<tr>
<td>detail_level</td>
<td>The level of detail shown. One of critical, debug, or info.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the service. One of running, error, or unknown.</td>
</tr>
<tr>
<td>status</td>
<td>An object with the service’s status details. Usually only relevant for error and unknown states.</td>
</tr>
<tr>
<td>active_alerts</td>
<td>An array of objects containing severity and a message about your replication from pglogical if you have replication enabled; otherwise, it’s an empty array.</td>
</tr>
</tbody>
</table>

For example, the response for a request to console services on port 4433 returns information about RBAC, the Activity Service, and the Classifier:

```json
{
  "activity-service": {
    "service_version": "2019.8.0.0",
    "service_status_version": 1,
    "detail_level": "info",
    "state": "running",
    "status": {
      "db_up": true,
      "db_pool": {
        "state": "ready"
      },
      "replication": {
```
Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service’s status is unknown or error
- 400 if a level parameter is set but is invalid (not critical, debug, or info)
GET /status/v1/services/<SERVICE_NAME>

Use the /services/<SERVICE_NAME> endpoint to retrieve the status of one particular PE service.

The content type for this endpoint is application/json; charset=utf-8.

Request format

This endpoint fetches the status for any one of the following services:

- activity-service (4433)
- classifier-service (4433)
- code-manager-service (8140)
- orchestrator-service (8143)
- server (8140)
- puppetdb-service (8081)

You must also specify the service's port in the request.

The request accepts the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>How thorough of a check to run. Set to critical, debug, or info (default).</td>
</tr>
<tr>
<td>timeout</td>
<td>Specified in seconds; defaults to 30.</td>
</tr>
</tbody>
</table>

For example, this request fetches info level status information about RBAC:

```
curl --insecure -H "X-Authentication: ALmWRLko6nJhi7bRQX4Qe-hOp07d4ZqDVhcf1rK81K8" \
   --header "Content-Type: application/json" \
   --request GET -G "https://puppet.example.com:4433/status/v1/services/rbac-service" \
   --data-url-encode '{"level": "info", "timeout": "30"}'
```

Response format

The response is a JSON object that lists details about the service, using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_version</td>
<td>Package version of the JAR file containing a given service.</td>
</tr>
<tr>
<td>service_status_version</td>
<td>The version of the API used to report the status of the service.</td>
</tr>
<tr>
<td>detail_level</td>
<td>The level of detail shown. One of critical, debug, or info.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the service. One of running, error, or unknown.</td>
</tr>
<tr>
<td>status</td>
<td>An object with the service’s status details. Usually only relevant for error and unknown states.</td>
</tr>
<tr>
<td>active_alerts</td>
<td>An array of objects containing severity and a message about your replication from pglogical if you have replication enabled; otherwise, it's an empty array.</td>
</tr>
</tbody>
</table>
For example, this response returned information about the RBAC service.

```json
{"rbac-service": {
  "service_version": "1.8.11-SNAPSHOT",
  "service_status_version": 1,
  "detail_level": "info",
  "state": "running",
  "status": {
    "activity_up": true,
    "db_up": true,
    "db_pool": { "state": "ready" },
    "replication": { "mode": "none", "status": "none" }
  },
  "active_alerts": []
}}
```

**Response codes**

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service’s status is unknown or error
- 400 if a level parameter is set but is invalid (not critical, debug, or info)
- 404 if no service named <SERVICE NAME> is found

**Status API: services plaintext endpoint**

The activity service plaintext endpoints are designed for load balancers that don't support any kind of JSON parsing or parameter setting. They return simple string bodies (either the state of the service in question or a simple error message) and a status code relevant to the status result.

**GET /status/v1/simple**

The /status/v1/simple returns a status that reflects all services the status service knows about.

The content type for this endpoint is text/plain; charset=utf-8.

**Query parameters**

No parameters are supported. Defaults to using the critical status level.

**Response codes**

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service’s status is unknown or error

**Possible responses**

The endpoint returns a status that reflects all services it knows about. It decides on what status to report using the following logic:

- running if and only if all services are running
- error if any service reports an error
- unknown if any service reports an unknown and no services report an error

**GET /status/v1/simple/<SERVICE NAME>**

The /status/v1/simple/<SERVICE NAME> endpoint returns the plaintext status of the specified service, such as rbac-service or classifier-service.

The content type for this endpoints is text/plain; charset=utf-8.
Query parameters
No parameters are supported. Defaults to using the critical status level.

Response codes
The server uses the following response codes:
- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error
- 404 if no service named <SERVICE NAME> is found

Possible responses
The endpoint returns a status that reflects all services it knows about. It decides on what status to report using the following logic:
- running if and only if all services are running
- error if any service reports an error
- unknown if any service reports an unknown and no services report an error
- not found: <SERVICENAME> if any service can't be found

Status API: metrics endpoint
Puppet Server is capable of tracking advanced metrics to give you additional insight into its performance and health.
The HTTPS metrics endpoints are available on port 8140 of the primary server:

```bash
uri="https://$(puppet config print server):8140/status/v1/services"
curl --insecure "$uri"
```

See Usage notes for curl examples for information about forming curl commands.

**Note:** These API endpoints are a tech preview. The metrics described here are returned only when passing the level=debug URL parameter, and the structure of the returned data might change in future versions.

These metrics fall into three categories:
- JRuby metrics (/status/v1/services/pe-jruby-metrics)
- HTTP route metrics (/status/v1/services/pe-master)
- Catalog compilation profiler metrics (/status/v1/services/pe-puppet-profiler)

All of these metrics reflect data for the lifetime of the current Puppet Server process and reset whenever the service is restarted. Any time-related metrics report milliseconds unless otherwise noted.

Like the standard status endpoints, the metrics endpoints return machine-consumable information about running services. This JSON response includes the same keys returned by a standard status endpoint request (see JSON endpoints). Each endpoint also returns additional keys in an experimental section.

**GET /status/v1/services/pe-jruby-metrics**
The /status/v1/services/pe-jruby-metrics endpoint returns JSON containing information about the JRuby pools from which Puppet Server fulfills agent requests.

You must query it at port 8140 and append the level=debug URL parameter.

Query parameters
No parameters are supported. Defaults to using the critical status level.
Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service’s status is unknown or error

Response keys

The metrics are returned in two subsections of the experimental section: jruby-pool-lock-status and metrics.

The response's experimental/jruby-pool-lock-status section contains the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>current-state</td>
<td>The state of the JRuby pool lock, which should be either :not-in-use (unlocked), :requested (waiting for lock), or :acquired (locked).</td>
</tr>
<tr>
<td>last-change-time</td>
<td>The date and time of the last current-state update, formatted as an ISO 8601 combined date and time in UTC.</td>
</tr>
</tbody>
</table>

The response's experimental/metrics section contains the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>average-borrow-time</td>
<td>The average amount of time a JRuby instance spends handling requests, calculated by dividing the total duration in milliseconds of the borrowed-instances value by the borrow-count value.</td>
</tr>
<tr>
<td>average-free-jrubies</td>
<td>The average number of JRuby instances that are not in use over the Puppet Server process’s lifetime.</td>
</tr>
<tr>
<td>average-lock-held-time</td>
<td>The average time the JRuby pool held a lock, starting when the value of jruby-pool-lock-status/current-state changed to :acquired. This time mostly represents file sync syncing code into the live codedir, and is calculated by dividing the total length of time that Puppet Server held the lock by the value of num-pool-locks.</td>
</tr>
<tr>
<td>average-lock-wait-time</td>
<td>The average time Puppet Server spent waiting to lock the JRuby pool, starting when the value of jruby-pool-lock-status/current-state changed to :requested. This time mostly represents how long Puppet Server takes to fulfill agent requests, and is calculated by dividing the total length of time that Puppet Server waits for locks by the value of num-pool-locks.</td>
</tr>
<tr>
<td>average-requested-jrubies</td>
<td>The average number of requests waiting on an available JRuby instance over the Puppet Server process’s lifetime.</td>
</tr>
<tr>
<td>average-wait-time</td>
<td>The average time Puppet Server spends waiting to reserve an instance from the JRuby pool, calculated by dividing the total duration in milliseconds of requested-instances by the requested-count value.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>borrow-count</td>
<td>The total number of JRuby instances that have been used.</td>
</tr>
<tr>
<td>borrow-retry-count</td>
<td>The total number of times that a borrow attempt failed and was retried, such as when the JRuby pool is flushed while a borrow attempt is pending.</td>
</tr>
<tr>
<td>borrow-timeout-count</td>
<td>The number of requests that were not served because they timed out while waiting for a JRuby instance.</td>
</tr>
<tr>
<td>borrowed-instances</td>
<td>A list of the JRuby instances currently in use, each reporting:</td>
</tr>
<tr>
<td></td>
<td>• duration-millis: The length of time that the instance has been running.</td>
</tr>
<tr>
<td></td>
<td>• reason/request: A hash of details about the request being served.</td>
</tr>
<tr>
<td></td>
<td>• request-method: The HTTP request method, such as POST, GET, PUT, or DELETE.</td>
</tr>
<tr>
<td></td>
<td>• route-id: The route being served. For routing metrics, see the HTTP metrics endpoint.</td>
</tr>
<tr>
<td></td>
<td>• uri: The request’s full URI.</td>
</tr>
<tr>
<td></td>
<td>• time: The time (in milliseconds since the Unix epoch) when the JRuby instance was borrowed.</td>
</tr>
<tr>
<td>num-free-jrubies</td>
<td>The number of JRuby instances in the pool that are ready to be used.</td>
</tr>
<tr>
<td>num-jrubies</td>
<td>The total number of JRuby instances.</td>
</tr>
<tr>
<td>num-pool-locks</td>
<td>The total number of times the JRuby pools have been locked.</td>
</tr>
<tr>
<td>requested-count</td>
<td>The number of JRuby instances borrowed, waiting, or that have timed out.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>requested-instances</td>
<td>A list of the requests waiting to be served, each reporting:</td>
</tr>
<tr>
<td></td>
<td>• duration-millis: The length of time the request has waited.</td>
</tr>
<tr>
<td></td>
<td>• reason/request: A hash of details about the waiting request.</td>
</tr>
<tr>
<td></td>
<td>• request-method: The HTTP request method, such as POST, GET, PUT, or DELETE.</td>
</tr>
<tr>
<td></td>
<td>• route-id: The route being served. For routing metrics, see the HTTP metrics endpoint.</td>
</tr>
<tr>
<td></td>
<td>• uri: The request’s full URI.</td>
</tr>
<tr>
<td></td>
<td>• time: The time (in milliseconds since the Unix epoch) when Puppet Server received the request.</td>
</tr>
<tr>
<td>return-count</td>
<td>The total number of JRuby instances that have been used.</td>
</tr>
</tbody>
</table>

For example:

```
"pe-jruby-metrics": {
   "detail_level": "debug",
   "service_status_version": 1,
   "service_version": "2.2.22",
   "state": "running",
   "status": {
      "experimental": {
         "jruby-pool-lock-status": {
            "current-state": ":not-in-use",
            "last-change-time": "2015-12-03T18:59:12.157Z"
         },
         "metrics": {
            "average-borrow-time": 292,
            "average-free-jrubies": 0.4716243097301104,
            "average-lock-held-time": 1451,
            "average-lock-wait-time": 0,
            "average-requested-jrubies": 0.21324752542875958,
            "average-wait-time": 156,
            "borrow-count": 639,
            "borrow-retry-count": 0,
            "borrow-timeout-count": 0,
            "borrowed-instances": [
               {  
                  "duration-millis": 3972,
                  "reason": {
                     "request": {
                        "request-method": "post",
                        "route-id": "puppet-v3-catalog-/*",
                        "uri": "/puppet/v3/catalog/hostame.example.com"
                     },
                     "time": 1448478371406
               }
            ]
         }
      }
   }
```

© 2022 Puppet, Inc., a Perforce company
GET /status/v1/services/pe-master

The /status/v1/services/pe-master endpoint returns JSON containing information about the routes that agents use to connect to this server.

You must query it at port 8140 and append the `level=debug` URL parameter.

**Query parameters**

No parameters are supported. Defaults to using the `critical` status level.

**Response codes**

The server uses the following response codes:

- 200 if and only if all services report a status of `running`
- 503 if any service's status is `unknown` or `error`

**Response keys**

The response's `experimental/http-metrics` section contains a list of routes, each containing the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>The total time Puppet Server spent processing requests for this route.</td>
</tr>
<tr>
<td>count</td>
<td>The total number of requests Puppet Server processed for this route.</td>
</tr>
<tr>
<td>mean</td>
<td>The average time Puppet Server spent on each request for this route, calculated by dividing the aggregate value by the count.</td>
</tr>
<tr>
<td>route-id</td>
<td>The route being served. The request returns a route with the special route-id of &quot;total&quot;, which represents the aggregate data for all requests along all routes.</td>
</tr>
</tbody>
</table>
Routes for newer versions of Puppet Enterprise and newer agents are prefixed with `puppet-v3`, while Puppet Enterprise 3 agents' routes are not. For example, a PE 2017.3 route-id might be `puppet-v3-report-/*/*`, while the equivalent PE 3 agent's route-id is `:environment-report-/*/*`.

For example:

```json
"pe-master": {
  {...},
  "status": {
    "experimental": {
      "http-metrics": [
        {
          "aggregate": 70668,
          "count": 234,
          "mean": 302,
          "route-id": "total"
        },
        {
          "aggregate": 28613,
          "count": 13,
          "mean": 2201,
          "route-id": "puppet-v3-catalog-/*/*"
        }
      ]
    }
  }
}
```

**GET /status/v1/services/pe-puppet-profiler**

The `/status/v1/services/pe-puppet-profiler` endpoint returns JSON containing statistics about catalog compilation. You can use this data to discover which functions or resources are consuming the most resources or are most frequently used.

You must query it at port 8140 and append the `level=debug` URL parameter.

The Puppet Server profiler is enabled by default, but if it has been disabled, this endpoint's metrics are not available. Instead, the endpoint returns the same keys returned by a standard status endpoint request and an empty `status` key.

**Query parameters**

No parameters are supported. Defaults to using the `critical` status level.

**Response codes**

The server uses the following response codes:

- 200 if and only if all services report a status of `running`
- 503 if any service's status is `unknown` or `error`

**Response keys**

If the profiler is enabled, the response returns two subsections in the `experimental` section:

- `experimental/function-metrics`, containing statistics about functions evaluated by Puppet Server when compiling catalogs.
- `experimental/resource-metrics`, containing statistics about resources declared in manifests compiled by Puppet Server.

Each function measured in the `function-metrics` section also has a `function` key containing the function's name, and each resource measured in the `resource-metrics` section has a `resource` key containing the resource's name.
The two sections otherwise share these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>The total time spent handling this function call or resource during catalog compilation.</td>
</tr>
<tr>
<td>count</td>
<td>The number of times Puppet Server has called the function or instantiated the resource during catalog compilation.</td>
</tr>
<tr>
<td>mean</td>
<td>The average time spent handling this function call or resource during catalog compilation, calculated by dividing the aggregate value by the count.</td>
</tr>
</tbody>
</table>

For example:

```
"pe-puppet-profiler": { 
  {...},
  "status": {
    "experimental": {
      "function-metrics": [
        {
          "aggregate": 1628,
          "count": 407,
          "function": "include",
          "mean": 4
        },
        {...},
        "resource-metrics": [
          {
            "aggregate": 3535,
            "count": 5,
            "mean": 707,
            "resource": "Class[Puppet_enterprise::Profile::Console]"
          },
          {...},
        ]
      }
    }
  }
}
```

### Metrics API

Puppet Enterprise includes an optional, enabled-by-default web endpoint for Java Management Extension (JMX) metrics, namely managed beans (MBeans).

These endpoints include:

- GET /metrics/v1/mbeans
- POST /metrics/v1/mbeans
- GET /metrics/v1/mbeans/<name>

**Note:** These API endpoints are a tech preview. The metrics described here are returned only when passing the `level=debug` URL parameter, and the structure of the returned data might change in future versions. To disable this endpoint, set `puppet_enterprise::master::puppetserver::metrics_webservice_enabled: false` in Hiera.
GET /metrics/v1/mbeans
The GET /metrics/v1/mbeans endpoint lists available MBeans.

Response keys
- The key is the name of a valid MBean.
- The value is a URI to use when requesting that MBean's attributes.

POST /metrics/v1/mbeans
The POST /metrics/v1/mbeans endpoint retrieves requested MBean metrics.

Query parameters
The query doesn't require any parameters, but the request body must contain a JSON object whose values are metric names, or a JSON array of metric names, or a JSON string containing a single metric's name.

For a list of metric names, make a GET request to /metrics/v1/mbeans.

Response keys
The response format, though always JSON, depends on the request format:
- Requests with a JSON object return a JSON object where the values of the original object are transformed into the MBeans' attributes for the metric names.
- Requests with a JSON array return a JSON array where the items of the original array are transformed into the MBeans' attributes for the metric names.
- Requests with a JSON string return the a JSON object of the Mbean's attributes for the given metric name.

GET /metrics/v1/mbeans/<name>
The GET /metrics/v1/mbeans/<name> endpoint reports on a single metric.

Query parameters
The query doesn't require any parameters, but the endpoint itself must correspond to one of the metrics returned by a GET request to /metrics/v1/mbeans.

Response keys
The endpoint's responses contain a JSON object mapping strings to values. The keys and values returned in the response vary based on the specified metric.

For example:
Use curl from localhost to request data on MBean memory usage:

```bash
curl 'http://localhost:8080/metrics/v1/mbeans/java.lang:type=Memory'
```

See Usage notes for curl examples for information about forming curl commands.

The response contains a JSON object representing the data:

```
{
  "ObjectPendingFinalizationCount" : 0,
  "HeapMemoryUsage" : {
    "committed" : 807403520,
    "init" : 268435456,
    "max" : 3817865216,
    "used" : 129257096
  },
  "NonHeapMemoryUsage" : {
    "committed" : 85590016,
  }
}
Managing nodes

Common node management tasks include adding and removing nodes from your deployment, grouping and classifying nodes, and running Puppet on nodes. You can also deploy code to nodes using an environment-based testing workflow or the roles and profiles method.

- Adding and removing agent nodes on page 452
  After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.
- Adding and removing agentless nodes on page 454
  Using the inventory, you can manage nodes, including devices such as network switches or firewalls, without installing the Puppet agent on them. The inventory stores node and device information securely.
• **How nodes are counted** on page 457
  Your *node count* is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your *bursting limit*. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

• **Running Puppet on nodes** on page 458
  Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

• **Grouping and classifying nodes** on page 460
  Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

• **Making changes to node groups** on page 468
  You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

• **Environment-based testing** on page 470
  An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

• **Preconfigured node groups** on page 472
  Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

• **Managing Windows nodes** on page 477
  You can use PE to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks using Forge modules.

• **Designing system configs: roles and profiles** on page 502
  Your typical goal with Puppet is to build complete system configurations, which manage all of the software, services, and configuration that you care about on a given system. The roles and profiles method can help keep complexity under control and make your code more reusable, reconfigurable, and refactorable.

• **Node classifier API v1** on page 528
  These are the endpoints for the node classifier v1 API.

• **Node classifier API v2** on page 578
  These are the endpoints for the node classifier v2 API.

• **Node inventory API** on page 581
  These are the endpoints for the node inventory v1 API.

### Adding and removing agent nodes

After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.

### Managing certificate signing requests

When you install a Puppet agent on a node, the agent automatically submits a certificate signing request (CSR) to the primary server. You must accept this request to bring before the node under PE management can be added your deployment. This allows Puppet to run on the node and enforce your configuration, which in turn adds node information to PuppetDB and makes the node available throughout the console.

You can approve certificate requests from the PE console or the command line. If DNS altnames are set up for agent nodes, you must approve the CSRs on the command line interface.

**Note:** Specific user permissions are required to manage certificate requests:

- To accept or reject CSRs in the console or on the command line, you need the permission **Certificate requests:**
  **Accept and reject.**

- To manage certificate requests in the console, you also need the permission **Console:** **View.**

### Related information

- **Installing agents** on page 223
You can install Puppet Enterprise agents on *nix, Windows, and macOS.

**Managing certificate signing requests in the console**
A certificate signing request appears in the console on the Certificates page in the Unsigned certificates tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

- To manage requests individually, click **Accept** or **Reject**.
- To manage the entire list of requests, click **Accept All** or **Reject All**. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

**Related information**
*Running Puppet on demand* on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

**Managing certificate signing requests on the command line**
You can view, approve, and reject node requests using the command line.

To view pending node requests on the command line:

```bash
$ sudo puppetserver ca list
```

To sign a pending request:

```bash
$ sudo puppetserver ca sign --certname <NAME>
```

**Note:** You can use the Puppet Server CA CLI to sign certificates with altnames or auth extensions by default.

**Remove agent nodes**
If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Purging a node:

- Removes the node from PuppetDB.
- Deletes the primary server’s information cache for the node.
- Makes the license available for another node.
- Makes the hostname available for another node.

**Note:** Removing a node doesn't uninstall the agent from the node.

1. On the agent node, stop the agent service: `service puppet stop`
2. On the primary server, purge the node: `puppet node purge <CERTNAME>`
   The node’s certificate is revoked, the certificate revocation list (CRL) is updated, and the node is removed from PuppetDB and the console. The license is now available for another node. The node can’t check in or re-register with PuppetDB on the next Puppet run.
3. If you have compilers, run Puppet on them: `puppet agent -t`
   The updated CRL is managed by Puppet and distributed to compilers.
4. Optional: If the node you’re removing was pinned to any node groups, you must manually unpin it from individual node groups or from all node groups using the `unpin-from-all` command endpoint.

**Related information**
*Uninstall infrastructure nodes* on page 254
The puppet-enterprise-uninstaller script is installed on the primary server. In order to uninstall, you must run the uninstaller on each infrastructure node.

**POST /v1/commands/unpin-from-all on page 563**

Use the /v1/commands/unpin-from-all to unpin specified nodes from all groups they’re pinned to. Nodes that are dynamically classified using rules aren’t affected.

### Uninstall *nix agents

The *nix agent package includes an uninstall script, which you can use when you're ready to retire a node.

1. On the agent node, run the uninstall script: `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`
2. Follow prompts to uninstall.
3. *(Optional)* If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean --certname <AGENT CERT NAME>`

### Uninstall Windows agents

To uninstall the agent from a Windows node, use the Windows **Add or Remove Programs** interface, or uninstall from the command line.

Uninstalling the agent removes the Puppet program directory, the agent service, and all related registry keys. The data directory remains intact, including all SSL keys. To completely remove Puppet from the system, manually delete the data directory.

1. Use the Windows **Add or Remove Programs** interface to remove the agent.

   Alternatively, you can uninstall from the command line if you have the original .msi file or know the product code of the installed MSI, for example: `msiexec /qn /norestart /x [puppet.msi|<PRODUCT_CODE>]`

2. *(Optional)* If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean --certname <AGENT CERT NAME>`

---

### Adding and removing agentless nodes

Using the inventory, you can manage nodes, including devices such as network switches or firewalls, without installing the Puppet agent on them. The inventory stores node and device information securely.

The inventory connects to agentless nodes through SSH or WinRM remote connections. The inventory uses transport definitions from installed device modules to connect to devices that can't have an agent installed on them.

After you add credentials to the inventory, authorized users can run tasks on these nodes and devices without re-entering credentials. On the **Tasks** page, these nodes and devices appear in the same list of targets as those that have agents installed.

### Add agentless nodes to the inventory

Add nodes over SSH or WinRM that will not or cannot have the Puppet agent installed to the inventory so you can run tasks on them.

**Before you begin**

Add classes to the **PE Master** node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.

Make sure you have the permission **Nodes: Add and delete connection information from inventory service**.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect over SSH or WinRM**.
3. Select a transport method.
   - SSH for *nix targets
   - WinRM for Windows targets

4. Enter target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.

5. Optional: Select additional target options. For example, to add a target port number, select Target Port from the drop-down list, enter the number, and click Add. For details, see Transport configuration options on page 455.

6. Click Add nodes.

After the nodes have been added to the inventory, they are added to PuppetDB, and you can view them from the Nodes page. Nodes in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory node's connection options, or to remove the node from inventory, go to the Connections tab on the node's details page.

Transport configuration options
A list of transport configuration options for SSH and WinRM transports.

Target options for SSH transport

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target port</td>
<td>Connection port. Default is 22.</td>
</tr>
<tr>
<td>Connection time-out in seconds</td>
<td>The length of time PE should wait when establishing connections.</td>
</tr>
<tr>
<td>Run as another user</td>
<td>After login, the user name to use for running commands.</td>
</tr>
<tr>
<td>Temporary directory</td>
<td>The directory to use when uploading temporary files to the target.</td>
</tr>
<tr>
<td>Sudo password</td>
<td>Password to use when changing users via run-as.</td>
</tr>
<tr>
<td>Process request as tty</td>
<td>Enable text terminal allocation.</td>
</tr>
</tbody>
</table>

Target options for WinRM transport

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target port</td>
<td>Connection port. Default is 5986, or 5985 if ssl: false</td>
</tr>
<tr>
<td>Connection time-out in seconds</td>
<td>The length of time PE should wait when establishing connections.</td>
</tr>
<tr>
<td>Temporary directory</td>
<td>The directory to use when uploading temporary files to the target.</td>
</tr>
<tr>
<td>Acceptable file extension</td>
<td>List of file extensions that are accepted for scripts or tasks. Scripts</td>
</tr>
<tr>
<td></td>
<td>with these file extensions rely on the target node's file type association</td>
</tr>
<tr>
<td></td>
<td>to run. For example, if Python is installed on the system, a .py script</td>
</tr>
<tr>
<td></td>
<td>should run with python.exe. The extensions .ps1, .rb, and .pp are always</td>
</tr>
<tr>
<td></td>
<td>allowed and run via hard-coded executables.</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
Add devices to the inventory

If you have installed modules for device transports in your production environment, you can add connections to those devices to your inventory. This lets you manage network devices such as switches and firewalls, and run Puppet and task jobs on them, just like other agentless nodes in your infrastructure.

**Before you begin**

Make sure you have the permission **Nodes: Add and delete connection information from inventory service.**

The connection details differ for each type of device transport, as defined in the module; see the device module's README for details.

⚠️ **CAUTION:** This initial implementation of network device configuration management is limited in how many device connections it can handle. Managing more than 100 devices can cause performance issues on the primary server. We plan to gather feedback from this release and improve the scaling and performance capabilities. In the meantime, avoid impacting primary server performance by limiting the number of network device connections you make to 100.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect network devices**.
3. Select a device type from the list of device transports that you have installed as modules in your production environment.
4. Enter the device certname and other connection details, as defined in the transport module. Mandatory fields are marked with an asterisk. See the module README file if you need more details or examples specific to the transport.
5. Click **Add node**.

After devices have been added to the inventory, they are added to PuppetDB, and you can view them from the **Nodes** page. Devices in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory device's connection options, or to remove the device from inventory, go to the **Connections** tab on the device's node details page.

Remove agentless nodes and devices from the inventory

Remove an agentless node or device connection from the inventory from the **Connections** tab on its details page.

**Before you begin**

Make sure you have the permission **Nodes: Add and delete connection information from inventory service.**

1. On the **Status** or **Nodes** page, find the node or device whose connection you want to remove, and click its name to open its details page.
2. Click **Connections**.
3. Click **Remove connection**. The exact name of the link varies depending on the connection type: **Remove SSH Connection**, **Remove WinRM connection**, or similar.
4. Confirm that you want to remove the connection.

When you remove a node or device connection from the inventory, PuppetDB marks it as expired after the standard node time-to-live (**node-ttl**) and then purges the node when it reaches its node-purge time-to-live limit (**node-purge-ttl**). At this point the node no longer appears in the console, and the node's license is available for use.

**Tip:** For more information about **node-ttl** and **node-purge-ttl** settings, see the PE docs for [database settings](#).

**Related information**

[Node inventory API on page 581](#)
These are the endpoints for the node inventory v1 API.

## How nodes are counted

Your *node count* is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your *bursting limit*. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

**Note:** *Node* in this context includes agent nodes, agentless nodes, primary servers, compilers, nodes running in *noop* mode, and nodes that have been purged but had prior activity within the same calendar month.

### Nodes included in the node count

The following nodes are included in your node count:

- Nodes with a report in PuppetDB during the calendar month.
- Nodes that have executed a Puppet run, task, or plan in the orchestrator, even if they do not have a report during the calendar month.

### Nodes not included in the node count

The following nodes are not included in your node count:

- Nodes that are in the inventory service but are not used with Puppet runs, tasks, or plans.
- Nodes that have been purged and have no reports or activity within the calendar month.

### Reaching the bursting limit

When you go above your node license's count limit, you enter what is called the *bursting limit*. The bursting limit lets you to exceed the number of nodes allowed under your license and enter a new threshold without extra charge. You are allowed to use your bursting limit on four consecutive or non-consecutive days per calendar month. Any higher usage beyond the four days will require you to either purge nodes or buy a license for more nodes.

The amount of time within your bursting limit does not matter for it to be counted as one day. For example, assuming you are licensed to use 1000 nodes and your bursting limit is 2000 nodes:

- If you use 1200 nodes for two hours one day, you have three days left to use your bursting limit that month.
- If you use 1900 nodes for 23 hours one day, you have three days left to use your bursting limit that month.
- If you use 1500 nodes for one hour each day for four days, you must contact your Puppet representative to buy a license for more nodes or purge some of your nodes until the next calendar month.

### When nodes are counted

PE tracks node counts daily from 12:00 midnight UTC to 12:00 midnight UTC.

The same time is used for the calendar month. For example, the month of September would include activity from 12:00 midnight UTC 01 September until 12:00 midnight UTC 01 October. After that point, the bursting limit restarts with a fresh four days in October.

### Viewing your node count

View your daily node count in the console by navigating to the **License** page and scrolling to the **Calendar month usage** section. This section also contains information about your subscription expiration date and license warnings, such as your license being expired or out of compliance.

To see daily node usage information on the command line, use the **Usage endpoints** on page 755.
Removing nodes

If you have unused nodes cluttering your inventory and are concerned about reaching your limit, read about removing them in Adding and removing agent nodes on page 452 and Adding and removing agentless nodes on page 454.

Related information
- Purchasing and installing a license key on page 222
- Your license must support the number of nodes that you want to manage with Puppet Enterprise.

Running Puppet on nodes

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

In a Puppet run, the primary server and agent nodes perform the following actions:

1. The agent node sends facts to the primary server and requests a catalog.
2. The primary server compiles and returns the agent’s catalog.
3. The agent applies the catalog by checking each resource the catalog describes. If it finds any resources that are not in the desired state, it makes the necessary changes.

**Note:** Puppet run behavior differs slightly if static catalogs are enabled.

Related information
- Static catalogs on page 312

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing `source` attributes pointing to `puppet://` locations on a node.

Running Puppet with the orchestrator

The Puppet orchestrator is a set of interactive tools used to deploy configuration changes when and how you want them. You can use the orchestrator to run Puppet from the console, command line, or API.

You can use the orchestrator to enforce change based on a:

- selection of nodes – from the console or the command line:
  ```
  puppet job run --nodes <COMMA-SEPARATED LIST OF NODE NAMES>
  ```
- PQL nodes query – from the console or the command line, for example:
  ```
  puppet job run --query 'nodes[certname] { facts {name = "operatingsystem" and value = "Debian" }}'
  ```
- A node group - from the console or the command line:
  ```
  puppet job run --node-group <node-group-id>
  ```

If you're putting together your own tools for running Puppet or want to enable CI workflows across your infrastructure, use the orchestrator API.

Related information
- Running Puppet on demand from the console on page 621
When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

**Running Puppet with SSH**

Run Puppet with SSH from an agent node.

SSH into the node and run `puppet agent --test` or `puppet agent -t`.

**Running Puppet from the console**

In the console, you can run Puppet from the node detail page for nodes that have an agent connection.

Run options include:

- **No-op** – Simulates changes without actually enforcing a new catalog. Nodes with `noop = true` in their `puppet.conf` files always run in no-op mode.
- **Debug** – Prints all messages available for use in debugging.
- **Trace** – Prints stack traces on some errors.
- **Evaltrace** – Shows a breakdown of the time taken for each step in the run.

When the run completes, the console displays the node’s run status.

**Note:** The Run Puppet button is not available if an agent does not have an active websocket session with the PCP broker, or if the node's connection method is SSH, WinRM, or a network device transport.

**Related information**

- **Node run statuses** on page 396
- The Status page displays the run status of each node following the most recent Puppet run. There are 10 possible run statuses.

**Activity logging on console Puppet runs**

When you initiate a Puppet run from the console, the Activity service logs the activity.

You can view activity on a single node by selecting the node, then clicking the Activity tab.

Alternatively, you can use the Activity Service API to retrieve activity information.

**Related information**

- **Activity service API** on page 382
- The activity service logs changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles.

**Troubleshooting Puppet run failures**

Puppet creates a View Report link for most failed runs, which you can use to access events and logs.

This table shows some errors that can occur when you attempt to run Puppet, and suggestions for troubleshooting.

<table>
<thead>
<tr>
<th>Error</th>
<th>Possible Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes could not be applied</td>
<td>Conflicting classes are a common cause. Check the log to get more detail.</td>
</tr>
<tr>
<td>Noop, changes could not be applied</td>
<td>Conflicting classes are a common cause. Check the log to get more detail.</td>
</tr>
<tr>
<td>Run already in progress</td>
<td>Occurs when a run is triggered in the command line or by another user, and you click Run.</td>
</tr>
<tr>
<td>Run request times out</td>
<td>Occurs if you click Run and the agent isn’t available.</td>
</tr>
</tbody>
</table>
Report request times out | Occurs when the report is not successfully stored in PuppetDB after the run completes.
Invalid response (such as a 500 error) | Your Puppet code might be have incorrect formatting.
Button is disabled and a run is not allowed. | The user has permission, but the agent is not responding.

### Grouping and classifying nodes

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

The main steps involved in classifying nodes are:

1. Create node groups.
2. Add nodes to groups, either manually or dynamically, with rules.
3. Assign classes to node groups.

Nodes can match the rules of many node groups. They receive classes, class parameters, and variables from all the node groups that they match.

### How node group inheritance works

Node groups exist in a hierarchy of parent and child relationships. Nodes inherit classes, class parameters and variables, and rules from all ancestor groups.

- Classes – If an ancestor node group has a class, all descendent node groups also have the class.
- Class parameters and variables – Descendent node groups inherit class parameters and variables from ancestors unless a different value is set for the parameter or variable in the descendent node group.
- Rules – A node group can only match nodes that all of its ancestors also match. Specifying rules in a child node group is a way of narrowing down the nodes in the parent node group to apply classes to a specific subset of nodes.

Because nodes can match multiple node groups from separate hierarchical lineages, it’s possible for two equal node groups to contribute conflicting values for variables and class parameters. Conflicting values cause a Puppet run on an agent to fail.

**Tip:** In the console, you can see how node groups are related on the **Node groups** page, which displays a hierarchical view of node groups. From the command line, you can use the group children endpoint to review group lineage.

**Related information**

GET `/v1/group-children/:id` on page 567
Use the `/v1/group-children/:id` endpoint to retrieve a specified group and its descendents.

### Best practices for classifying node groups

To organize node groups, start with the high-level functional groups that reflect the business requirements of your organization, and work down to smaller segments within those groups.

For example, if a large portion of your infrastructure consists of web servers, create a node group called `web servers` and add any classes that need to be applied to all web servers.

Next, identify subsets of web servers that have common characteristics but differ from other subsets. For example, you might have production web servers and development web servers. So, create a `dev web` child node group under the `web servers` node group. Nodes that match the `dev web` node group get all of the classes in the parent node group in addition to the classes assigned to the `dev web` node group.
Environment versus classification node groups

Environment node groups assign environments to nodes, such as test, development, or production. A node can belong to only one environment node group. If a node is added to more than one environment group, classification errors occur. See this classification conflict article for more information.

An environment node group:
- Must be used in conjunction with a corresponding Git branch in a control repo you want to use for targeted code deploys. The Git branch and environment group must share the same name.
- Must be a child of the All Environments node group
- Must not include child groups, except any one-time run exception subgroups used for canary testing
- Must not include classes or configuration data.

Classification node groups assign classification data to nodes, including classes, parameters, and variables. A node can belong in more than one classification group.

A classification node group:
- Must be a child of All Nodes or another classification group.
- Must not be specified as an environment group in the group metadata.

Related information
Environment-based testing on page 470
An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

Create node groups

Use the console to create node groups to assign either an environment or classification.

Create environment node groups
Create custom environment node groups so that you can target deployment of Puppet code.

1. In the console, click Node groups, and click Add group.
2. Specify options for the new node group and then click Add.
   - Parent name – Select the top-level environment node group in your hierarchy. If you're using default environment node groups, this might be Production environment or All environments. Every environment node group you add must be a descendant of the top-level environment node group.
   - Group name – Enter a name that describes the role of this environment node group, for example, Test environment.
   - Environment – Select the environment that you want to assign to nodes that match this node group.
   - Environment group - Select this option.

You can now add nodes to your environment node group to control which environment each node belongs to.

Related information
Environment-based testing on page 470
An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

Create classification node groups
Create classification node groups to assign classification data to nodes.

1. In the console, click Node groups, and click Add group.
2. Specify options for the new node group and then click Add.

- **Parent name** – Select the name of the classification node group that you want to set as the parent to this node group. Classification node groups inherit classes, parameters, and variables from their parent node group. By default, the parent node group is the All Nodes node group.
- **Group name** – Enter a name that describes the role of this classification node group, for example, Web Servers.
- **Environment** – Specify an environment to limit the classes and parameters available for selection in this node group.

Note: Specifying an environment in a classification node group does not assign an environment to any nodes, as it does in an environment node group.

- **Environment group** – Do not select this option.

You can now add nodes to your classification node group dynamically or statically.

### Add nodes to a node group

There are two ways to add nodes to a node group.

- Individually pin nodes to the node group (static)
- Create rules that match node facts (dynamic)

#### Statically add nodes to a node group

If you have a node that needs to be in a node group regardless of the rules specified for that node group, you can pin the node to the node group.

A pinned node remains in the node group until you manually remove it. Adding a pinned node essentially creates the rule `<the certname of your node> = <the certname>`, and includes this rule along with the other fact-based rules.

1. In the console, click **Node groups**, and then find the node group that you want to pin a node to and select it.
2. On the **Rules** tab, in the pinned nodes section, enter the certname of the node.
3. Click **Pin node**, and then commit changes.

#### Dynamically add nodes to a node group

Rules are the most powerful and scalable way to include nodes in a node group. You can create rules in a node group that are used to match node facts.

When nodes match the rules in a node group, they’re classified with all of the classification data (classes, parameters, and variables) for the node group.

When nodes no longer match the rules of a node group, the classification data for that node group no longer applies to the node.

1. In the console, click **Node Groups**, and then find the node group that you want to add the rule to and select it.
2. On the **Rules** tab, specify rules for the fact, then click **Add rule**.
3. (Optional) Repeat step 2 as needed to add more rules.

   **Tip:** If the node group includes multiple rules, be sure to specify whether **Nodes must match all rules** or **Nodes can match any rule**.

4. Commit changes.
**Writing node group rules**

To dynamically assign nodes to a group, you must specify rules based on node facts. Use this reference to fill out the **Rules** tab for node groups.

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>Specifies the fact used to match nodes. Select from the list of known facts, or enter part of a string to view fuzzy matches. To use structured or trusted facts, select the initial value from the dropdown list, then type the rest of the fact.</td>
</tr>
<tr>
<td></td>
<td>• To descend into a hash, use dots (&quot;.&quot;)) to designate path segments, such as <code>os.release.major</code> or <code>trusted.certname</code>.</td>
</tr>
<tr>
<td></td>
<td>• To specify an item in an array, surround the numerical index of the item in square brackets, such as <code>processors.models[0]</code> or <code>mountpoints./.options[0]</code>.</td>
</tr>
<tr>
<td></td>
<td>• To identify path segments that contain dots, periods, dashes, spaces, or UTF characters, surround the segment with single or double quotes, such as <code>trusted.extensions.&quot;1.3.6.1.4.1.34380.1.2.1&quot;</code>.</td>
</tr>
<tr>
<td></td>
<td>• To use trusted extension short names, append the short name after a second dot, such as <code>trusted.extensions.pp_role</code>.</td>
</tr>
</tbody>
</table>

**Tip:** Structured and trusted facts don't provide type-ahead suggestions beyond the top-level name key, and the facts aren't verified when entered. After adding a rule for a structured or trusted fact, review the number of matching nodes to verify that the fact was entered correctly.
<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Describes the relationship between the fact and value. Operators include:</td>
</tr>
<tr>
<td></td>
<td>• =  — is</td>
</tr>
<tr>
<td></td>
<td>• !=  — is not</td>
</tr>
<tr>
<td></td>
<td>• ~  — matches regex</td>
</tr>
<tr>
<td></td>
<td>• !~  — does not match regex</td>
</tr>
<tr>
<td></td>
<td>• &gt;  — greater than</td>
</tr>
<tr>
<td></td>
<td>• &gt;=  — greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>• &lt;  — less than</td>
</tr>
<tr>
<td></td>
<td>• &lt;=  — less than or equal to</td>
</tr>
<tr>
<td></td>
<td>The numeric operators &gt;, &gt;=, &lt;, and &lt;= can be used only with facts that have a numeric value.</td>
</tr>
<tr>
<td></td>
<td>To match highly specific node facts, use ~ or !~ with a regular expression for Value.</td>
</tr>
<tr>
<td>Value</td>
<td>Specifies the value associated with the fact.</td>
</tr>
</tbody>
</table>

**Using structured and trusted facts for node group rules**

Structured facts group a set of related facts, whereas trusted facts are a specific type of structured fact.

**Structured facts** group a set of related facts in the form of a hash or array. For example, the structured fact `os` includes multiple independent facts about the operating system, including architecture, family, and release. In the console, when you view facts about a node, you can differentiate structured facts because they’re surrounded by curly braces.

**Trusted facts** are a specific type of structured fact where the facts are immutable and extracted from a node’s certificate. Because they can’t be changed or overridden, trusted facts enhance security by verifying a node’s identity before sending sensitive data in its catalog.

You can use structured and trusted facts in the console to dynamically add nodes to groups.

**Note:** If you’re using trusted facts to specify certificate extensions, in order for nodes to match to rules correctly, you must use short names for Puppet registered IDs and numeric IDs for private extensions. Numeric IDs are required for private extensions whether or not you specify a short name in the `custom_trusted_oid_mapping.yaml` file.

**Declare classes**

Classes are the blocks of Puppet code used to configure nodes and assign resources to them.

**Before you begin**

The class that you want to apply must exist in an installed module. You can download modules from the Forge or create your own module.

1. In the console, click **Node groups**, and then find the node group that you want to add the class to and select it.
2. On the **Classes** tab, select the class to add.

   The **Add new class** field suggests classes that the primary server knows about and that are available in the environment set for the node group.

3. Click **Add class** and then commit changes.

   **Note:** Classes don’t appear in the class list until they’re retrieved from the primary server and the environment cache is refreshed. By default, both of these actions occur every three minutes. To override the default refresh period and force the node classifier to retrieve the classes from the primary server immediately, click the **Refresh** button.

---

**Enable data editing in the console**

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

1. On your primary server, edit `/etc/puppetlabs/puppet/hiera.yaml` to add:

   ```yaml
   hierarchy:
   - name: "Classifier Configuration Data"
     data_hash: classifier_data
   
   Place any additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same hierarchy key, preferably below the **Classifier Configuration Data** entry.

   **Note:** If you enable data editing in the console, add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

2. If your environment is configured for high availability or has compilers, update `hiera.yaml` on your replica and compilers, respectively.

---

**Define data used by node groups**

The console offers multiple ways to specify data used in your manifests.

- **Configuration data** — Specify values through automatic parameter lookup.
- **Parameters** — Specify resource-style values used by a declared class.
- **Variables** — Specify values to make available in Puppet code as top-scope variables.

**Set configuration data**

Configuration data set in the console is used for automatic parameter lookup, the same way that Hiera data is used. Console configuration data takes precedence over Hiera data, but you can combine data from both sources to configure nodes.

**Tip:** In most cases, setting configuration data in Hiera is the more scalable and consistent method, but there are some cases where the console is preferable. Use the console to set configuration data if:

- You want to override Hiera data. Data set in the console overrides Hiera data when configured as recommended.
- You want to give someone access to set or change data and they don’t have the skill set to do it in Hiera.
- You simply prefer the console user interface.

**Important:** If your installation includes a disaster recovery replica, make sure you’ve correctly enabled data editing in the console for both your primary server and replica.

1. In the console, click **Node groups**, then find the node group that you want to add configuration data to and select it.
2. On the **Configuration data** tab, specify a **Class** and select a **Parameter** to add.

   You can select from existing classes and parameters in the node group’s environment, or you can specify free-form values. Classes aren’t validated, but any class you specify must be present in the node’s catalog at runtime in order for the parameter value to be applied.

   When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. Optional: Change the default parameter **Value**.

**Related information**

*Enable data editing in the console* on page 302

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your **hiera.yaml** file.

**Set parameters**

Parameters are declared resource-style, which means they can be used to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

1. In the console, click **Node groups**, and then find the node group that you want to add a parameter to and select it.
2. On the **Classes** tab, select the class you want to modify and the **Parameter** to add.

   The **Parameter** drop-down list shows all of the parameters that are available for that class in the node group’s environment. When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. (Optional) Change the default **Value**.

**Set variables**

Variables set in the console become top-scope variables available to all Puppet manifests.

1. In the console, click **Node groups**, and then find the node group that you want to set a variable for and select it.
2. On the **Variables** tab, enter options for the variable:
   - **Key** – Enter the name of the variable.
   - **Value** – Enter the value that you want to assign to the variable.
3. Click **Add variable**, and then commit changes.

**Tips for specifying parameter and variable values**

Parameters and variables can be structured as JSON. If they can’t be parsed as JSON, they're treated as strings.

Parameters and variables can be specified using these data types and syntax:

- **Strings** (for example, "centos")
  - Variable-style syntax, which interpolates the result of referencing a fact (for example, "I live at $ipaddress.")
  - Expression-style syntax, which interpolates the result of evaluating the embedded expression (for example, ${$os"release"})

**Note:** Strings must be double-quoted, because single quotes aren't valid JSON.

**Tip:** To enter a value in the console that contains a literal dollar sign, like a password hash — for example, $1$nnkkFwEc$saffPMXYaUVKZrDV4FLCm0/ — escape each dollar sign with a backslash to disable interpolation.

- **Booleans** (for example, true or false)
- **Numbers** (for example, 123)
• Hashes (for example, "a": 1))

**Note**: Hashes must use colons rather than hash rockets.

• Arrays (for example, ["1", "2.3"])

**Variable-style syntax**

Variable-style syntax uses a dollar sign ($) followed by a Puppet fact name.

Example: "I live at $ipaddress"

Variable-style syntax is interpolated as the value of the fact. For example, $ipaddress resolves to the value of the ipaddress fact.

**Important**: The endpoint variable $pe_node_groups cannot be interpolated when used as a classifier in class variable values.

Indexing cannot be used in variable-style syntax because the indices are treated as part of the string literal. For example, given the following fact: processors => {"count" => 4, "physicalcount" => 1}, if you use variable-style syntax to specify $processors[count], the value of the processors fact is interpolated but it is followed by a literal "[count]". After interpolation, this example becomes {"count" => 4, "physicalcount" => 1}[count].

**Note**: Do not use the :: top-level scope indication because the console is not aware of Puppet variable scope.

**Expression-style syntax**

Use expression-style syntax when you need to index into a fact (${$os[release]}), refer to trusted facts ("My name is ${trusted[certname]}"), or delimit fact names from strings ("My ${os} release").

The following is an example of using expression-style syntax to access the full release number of an operating system:

```
${$os"release"}
```

Expression-style syntax uses the following elements in order:

• an initial dollar sign and curly brace (${)
• a legal Puppet fact name preceded by an optional dollar sign
• any number of index expressions (the quotations around indices are optional but are required if the index string contains spaces or square brackets)
• a closing curly brace (})

Indices in expression-style syntax can be used to access individual fields of structured facts, or to refer to trusted facts. Use strings in an index if you want to access the keys of a hashmap. If you want to access a particular item or character in an array or string based on the order in which it is listed, you can use an integer (zero-indexed).

Examples of legal expression-style interpolation:

• ${os}
• ${os}
• ${os["release"]}
• ${os["release"]}
• ${os["release"]}
• ${os[2]} (accesses the value of the third (zero-indexed) key-value pair in the os hash)
• ${os["release"]} (accesses the value of the third key-value pair in the release hash)

In the console, an index can be only simple string literals or decimal integer literals. An index cannot include variables or operations (such as string concatenation or integer arithmetic).
Examples of illegal expression-style interpolation:

- ${::os}
- ${os[$release]}
- ${os[0xff]}
- ${os[6/3]}
- ${os[$family + $release]}
- ${os + $release}

**Trusted facts**

Trusted facts are considered to be keys of a hashmap called trusted. This means that all trusted facts must be interpolated using expression-style syntax. For example, the certname trusted fact would be expressed like this: "My name is ${trusted[certname]}". Any trusted facts that are themselves structured facts can have further index expressions to access individual fields of that trusted fact.

**Note:** Regular expressions, resource references, and other keywords (such as ‘undef’) are not supported.

**View nodes in a node group**

To view all nodes that currently match the rules specified for a node group:

1. In the console, click **Node groups**, and then find the node group that you want to view and select it.
2. Click **Matching nodes**.

You see the number of nodes that match the node group’s rules, along with a list of the names of matching nodes. This is based on the facts collected during the node’s last Puppet run. The matching nodes list is updated as rules are added, deleted, and edited. Nodes must match rules in ancestor node groups as well as the rules of the current node group in order to be considered a matching node.

**Making changes to node groups**

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

**Edit or remove node groups**

You can change the name, description, parent, environment, or environment group setting for node groups, or you can delete node groups that have no children.

1. In the console, click **Node groups**, and select a node group.
2. At the top right of the page, select an option.
   - **Edit node group metadata** — Enables edit mode so you can modify the node group’s metadata as needed.
   - **Remove node group** — Removes the node group. You’re prompted to confirm the removal.
3. Commit changes.

**Remove nodes from a node group**

To remove dynamically-assigned nodes from a node group, edit or delete the applicable rule. To remove statically-assigned nodes from a node group, unpin them from the group.

**Note:** When a node no longer matches the rules of a node group, it is no longer classified with the classes assigned in that node group. However, the resources that were installed by those classes are not removed from the node. For example, if a node group has the apache class that installs the Apache package on matching nodes, the Apache package is not removed from the node even when the node no longer matches the node group rules.

1. In the console, click **Node groups**, and select a node group.
2. On the Rules tab, select the option appropriate for the type of node.
   - Dynamically-assigned nodes — In the Fact table, click Remove to remove an individual rule, or click Remove all rules to delete all rules for the node group.
   - Statically-assigned nodes — In the Certname table, click Unpin to unpin an individual node, or click Unpin all pinned nodes to unpin all nodes from the node group.

   **Tip:** To unpin a node from all groups it’s pinned to, use the unpin-from-all command endpoint.

3. Commit changes.

**Related information**

POST /v1/commands/unpin-from-all on page 563

Use the /v1/commands/unpin-from-all to unpin specified nodes from all groups they’re pinned to. Nodes that are dynamically classified using rules aren’t affected.

**Remove classes from a node group**

Make changes to your node group by removing classes.

**Note:** If a class appears in a node group list but is crossed out, the class has been deleted from Puppet.

1. In the console, click Node groups, and select a node group.
2. On the Classes tab, click Remove this class to remove an individual class, or click Remove all classes to remove all classes from the node group.
3. Commit changes.

**Edit or remove parameters**

Make changes to your node group by editing or deleting the parameters of a class.

1. In the console, click Node groups, and select a node group.
2. On the Classes tab, select an option for the class and parameter you want to edit.
   - Edit — Enables edit mode so you can modify the parameter as needed.
   - Remove — Removes the parameter.
3. Commit changes.

**Edit or remove variables**

Make changes to your node group by editing or removing variables.

1. In the console, click Node groups, and select a node group.
2. On the Variables tab, select an option.
   - Edit — Enables edit mode so you can modify the variable as needed.
   - Remove — Removes the variable.
   - Remove all variables — Removes all variables from the node group.
3. Commit changes.
Environment-based testing

An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

1. **Root All Nodes** node group.

2. **All Environments** parent node group. All environment node groups must be children of this group. The Puppet environment assigned to this group is the default Puppet environment used for nodes without an assigned or matched default environment.

   **Tip:** If this group doesn't exist, you should create it.

3. **Environment** node groups. Environment node groups are used to specify the Puppet environment that nodes belong to. Nodes can belong to no more than one environment node group. The group it belongs to determines its default environment.

   Each environment node group must meet these conditions:
   - Must be a direct child of the **All Environments** node group
   - Must not include child groups, except any one-time run exception subgroups used for canary testing
   - Must be specified as an environment group in the group metadata
   - Must not include classes or configuration data
4. Optional one-time run exception group. This group acts as a gatekeeper, and when present, permits nodes in the parent environment group to temporarily use Puppet environments other than their normal default environment.

5. PE Infrastructure node group and its children. These are built-in classification node groups used to manage infrastructure nodes. Don't modify these groups except when following official documentation or support instructions.

6. Classification node groups. Classification node groups are used to apply classes and configuration data to nodes. Nodes can match more than one classification node group.

   Each classification node group must meet these conditions:
   
   • Must be a child of All Nodes or another classification group
   • Must not be specified as an environment group in the group metadata

Test and promote a parameter

Test and promote a parameter when using an environment-based testing workflow.

1. Create a classification node group with the test environment that is a child of a classification group that uses the production environment.
2. In the child group, set a test parameter. The test parameter overrides the value set by the parent group using the production environment.
3. If you're satisfied with your test results, manually change the parameter in the parent group.
4. Delete the child test group.

Test and promote a class

Test and promote a class when using an environment-based testing workflow.

1. Create a classification node group with the test environment that is a child of a classification group that uses the production environment.
   The node classifier validates your parameters against the test environment.
2. Add the class you would like to test.
3. If you're satisfied with your test results, promote your code to production and manually add the class in the parent node group.
4. Delete the child test group.

Test code with canary nodes and alternate Puppet environments

You can test new code using one-time Puppet agent runs on select canary nodes.

Before you begin

You must have configured an environment group other than All Environments in which to test code.

To enable the canary workflow, configure a one-time run exception child group for each environment group in which you want to test code. This group matches nodes on the fly when you run Puppet with the environment specified.

In this example, we configure a one-time run exception environment group as a child of the Development environment. You can create a similar group for any environment.

1. Create the one-time run exception group with these options:
   
   • Name — Development one-time run exception
   • Parent — Development environment
   • Environment — agent-specified
   • Environment group — checked
2. Create a rule to match nodes to this group if they request a Puppet environment other than their default.
   - **Fact** — `agent_specified_environment`
     
     **Tip:** This fact name doesn't autocomplete.
   - **Operator** — `~`
   - **Value** — `.+`

   This rule matches any node from the parent group that requests the use of a non-default environment using either
   the `--environment` flag on the command line or the `environment` setting in `puppet.conf`.

3. On any node in the **Development environment** group, run `puppet agent -t --environment <ENVNAME>`, where `<ENVNAME>` is the name of the Puppet environment that contains your test code.

   If you're using Code Manager and a Git workflow, `<ENVNAME>` is the name of your Git development or feature branch.

   During this Puppet run, the agent sets the `agent_specified_environment` value to `<ENVNAME>`. The
   Development one-time run exception group matches the node and permits it to use the requested
   environment.

---

**Sample rules for one-time run exception groups**

These examples show several ways to configure rules for one-time run exception child groups. Where multiple rules
are listed, combine the rules by specifying that nodes must match all rules.

<table>
<thead>
<tr>
<th>Testing scenario</th>
<th>Fact</th>
<th>Operator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permit any node in the parent environment group to use any Puppet environment</td>
<td><code>agent_specified_environment</code></td>
<td><code>.+</code></td>
<td></td>
</tr>
<tr>
<td>Permit RHEL nodes in the parent environment group to use any Puppet environment</td>
<td><code>agent_specified_environment</code></td>
<td><code>.+</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>facts.os.family</code></td>
<td><code>=</code></td>
<td>RedHat</td>
</tr>
<tr>
<td>Permit any nodes in the parent environment group to use any Puppet environment</td>
<td><code>agent_specified_environment</code></td>
<td><code>.+</code></td>
<td></td>
</tr>
<tr>
<td>except production</td>
<td><code>agent_specified_environment</code></td>
<td><code>!=</code></td>
<td>production</td>
</tr>
<tr>
<td>Permit any nodes in the parent environment group to use any Puppet environment that</td>
<td><code>agent_specified_environment</code></td>
<td><code>.+</code></td>
<td></td>
</tr>
<tr>
<td>begins with the prefix <code>feature_</code></td>
<td><code>agent_specified_environment</code></td>
<td><code>^feature_.+</code></td>
<td></td>
</tr>
</tbody>
</table>

---

**Preconfigured node groups**

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

**All Nodes node group**

This node group is at the top of the hierarchy tree. All other node groups stem from this node group.

**Classes**

No default classes. Avoid adding classes to this node group.
Matching nodes
All nodes.

Notes
You can't modify the preconfigured rule that matches all nodes.

Infrastructure node groups
Infrastructure node groups are used to manage PE.

**Important:** Don't make changes to infrastructure node groups other than pinning new nodes for documented functions, like creating compilers. If you want to add custom classifications to infrastructure nodes, create new child groups and apply classification there.

**PE Infrastructure node group**
This node group is the parent to all other infrastructure node groups.

The **PE Infrastructure** node group contains data such as the hostnames and ports of various services and database info (except for passwords).

It's very important to correctly configure the `puppet_enterprise` class in this node group. The parameters set in this class affect the behavior of all other preconfigured node groups that use classes starting with `puppet_enterprise::profile`. Incorrect configuration of this class could potentially cause a service outage.

⚠️ **CAUTION:** Never remove the **PE Infrastructure** node group. Removing the **PE Infrastructure** node group disrupts communication between all of your **PE Infrastructure** nodes.

### Classes
`puppet_enterprise` — sets the default parameters for child node groups

### Matching nodes
Nodes are not pinned to this node group. The **PE Infrastructure** node group is the parent to other infrastructure node groups, such as **PE Master**, and is used only to set classification that all child node groups inherit. Never pin nodes directly to this node group.

These are the parameters for the `puppet_enterprise` class, where `<YOUR HOST>` is your primary server certname. You can find the certname with `puppet config print certname`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>database_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
<tr>
<td>puppetdb_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
<tr>
<td>database_port</td>
<td>&quot;&lt;YOUR PORT NUMBER&gt;&quot;</td>
</tr>
<tr>
<td></td>
<td>Required only if you changed the port number from the default 5432.</td>
</tr>
<tr>
<td>database_ssl</td>
<td>true if you're using the PE-installed PostgreSQL, and false if you're using your own PostgreSQL.</td>
</tr>
<tr>
<td>puppet_master_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
<tr>
<td>certificate_authority_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>console_port</td>
<td>&quot;&lt;YOUR PORT NUMBER&gt;&quot;</td>
</tr>
<tr>
<td></td>
<td>Required only if you changed the port number from the default 443.)</td>
</tr>
<tr>
<td>puppetdb_database_name</td>
<td>&quot;pe-puppetdb&quot;</td>
</tr>
<tr>
<td>puppetdb_database_user</td>
<td>&quot;pe-puppetdb&quot;</td>
</tr>
<tr>
<td>puppetdb_port</td>
<td>&quot;&lt;YOUR PORT NUMBER&gt;&quot;</td>
</tr>
<tr>
<td></td>
<td>Required only if you changed the port number from the default 8081.)</td>
</tr>
<tr>
<td>console_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
<tr>
<td>pcp_broker_host</td>
<td>&quot;&lt;YOUR HOST&gt;&quot;</td>
</tr>
</tbody>
</table>

### PE Certificate Authority node group
This node group is used to manage the certificate authority.

**Classes**

- `puppet_enterprise::profile::certificate_authority` — manages the certificate authority on the primary server

**Matching nodes**

- On a new install, the primary server is pinned to this node group.

**Notes**

- Don't add additional nodes to this node group.

### PE Master node group
This node group is used to manage the primary server.

**Classes**

- `puppet_enterprise::profile::master` — manages the primary server service

**Matching nodes**

- On a new install, the primary server is pinned to this node group.

### PE Compiler node group
This node group is a subset of the **PE Master** node group used to manage compilers running the PuppetDB service.

**Classes**

- `puppet_enterprise::profile::master` — manages the primary server service
- `puppet_enterprise::profile::puppetdb` — manages the PuppetDB service

**Matching nodes**

- Compilers running the PuppetDB service are automatically added to this node group.

**Notes**

- Don't add additional nodes to this node group.
**PE Orchestrator node group**
This node group is used to manage the PE orchestration services configuration, which includes things like task concurrency limits, the PCP broker timeout, and how many JRubies can run in the orchestrator at one time.

**Classes**
- puppet-enterprise::profile::orchestrator — manages PE orchestration services

**Matching nodes**
- On a new install, the primary server is pinned to this node group.

**Notes**
- Don’t add additional nodes to this node group.

**PE PuppetDB node group**
This node group is used to manage nodes running the PuppetDB service. If the node is also serving as a compiler, it's instead classified in the PE Compiler node group.

**Classes**
- puppet-enterprise::profile::puppetdb — manages the PuppetDB service

**Matching nodes**
- PuppetDB nodes that aren't functioning as compilers are pinned to this node group.

**Notes**
- Don’t add additional nodes to this node group.

**PE Console node group**
This node group is used to manage the console.

**Classes**
- puppet-enterprise::profile::console — manages the console, node classifier, and RBAC
- puppet-enterprise::license — manages the PE license file for the status indicator

**Matching nodes**
- On a new install, the console server node is pinned to this node group.

**Notes**
- Don’t add additional nodes to this node group.

**PE Agent node group**
This node group is used to manage the configuration of agents.

**Classes**
- puppet-enterprise::profile::agent — manages your agent configuration

**Matching nodes**
- All managed nodes are pinned to this node group by default.

**PE Infrastructure Agent node group**
This node group is a subset of the PE Agent node group used to manage infrastructure-specific overrides.

**Classes**
- puppet-enterprise::profile::agent — manages your agent configuration

**Matching nodes**
- All nodes used to run your Puppet infrastructure and managed by the PE installer are pinned to this node group by default, including the primary server, PuppetDB, console, and compilers.
Notes
You might want to manually pin to this group any additional nodes used to run your infrastructure, such as compiler load balancer nodes. Pinning a compiler load balancer node to this group allows it to receive its catalog from the primary server, rather than the compiler, which helps ensure availability.

PE Database node group
This node group is used to manage the PostgreSQL service.

Classes
- puppet_enterprise::profile::database — manages the PE-PostgreSQL service

Matching nodes
The node specified as puppet_enterprise::database_host is pinned to this group. By default, the database host is the PuppetDB server node.

Notes
Don't add additional nodes to this node group.

PE Patch Management node group
This is a parent node group for nodes under patch management. Create child node groups based on your needs.

Classes
- pe_patch — enables patching on nodes.

Matching nodes
There are no nodes pinned to this group. PE Patch Management is a parent group for node groups under patch management. You can create node groups with unique configurations based on your patching needs.

Notes
Don't add additional nodes to this node group, only add node groups.

Related information
Create a node group for nodes under patch management on page 594
Create a node group for nodes you want to patch in PE and add nodes to it. For example, create a node group for testing Windows and *nix patches prior to rolling out patches to other node groups. The PE Patch Management parent node group has the pe_patch class assigned to it and is in the console by default.

Environment node groups
Environment node groups are used only to set environments. They cannot contain any classification.

Preconfigured environment node groups differ depending on the version of PE you're on and you can customize environment groups as needed for your ecosystem. If you upgrade from an older version of PE, your environment node groups stay the same as they were in the older version.
Managing Windows nodes

You can use PE to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks using Forge modules.

Basic tasks and concepts in Windows

This section is meant to help familiarize you with several common tasks used in Puppet Enterprise (PE) with Windows agents, and explain the concepts and reasons behind performing them.

Practice tasks

In other locations in the documentation, these can be found as steps in tasks, but they are not explained as thoroughly.

Write a simple manifest

Puppet manifest files are lists of resources that have a unique title and a set of named attributes that describe the desired state.

Before you begin

You need a text editor, for example Visual Studio Code (VS Code), to create manifest files. Puppet has an extension for VS Code that supports syntax highlighting of the Puppet language. Editors like Notepad++ or Notepad won’t highlight Puppet syntax, but can also be used to create manifests.

Manifest files are written in Puppet code, a domain specific language (DSL) that defines the desired state of resources on a system, such as files, users, and packages. Puppet compiles these text-based manifests into catalogs, and uses those to apply configuration changes.

1. Create a file named `file.pp` and save it in `c:\myfiles\`
2. With your text editor of choice, add the following text to the file:

   ```puppet
   file { 'c:\Temp\foo.txt':
     ensure   => present,
     content  => 'This is some text in my file'
   }
   ```

Note the following details in this file resource example:

- Puppet uses a basic syntax of `type { title: }`, where `type` is the resource type — in this case it’s `file`.
- The resource title (the value before the :) is `C:\Temp\foo.txt`. The file resource uses the title to determine where to create the file on disk. A resource title must always be unique within a given manifest.
- The `ensure` parameter is set to `present` to create the file on disk, if it's not already present. For `file` type resources, you can also use the value `absent`, which removes the file from disk if it exists.
- The `content` parameter is set to `This is some text in my file`, which writes that value to the file.

Validate your manifest with `puppet parser validate`

You can validate that a manifest's syntax is correct by using the command `puppet parser validate`

1. Check your syntax by entering `puppet parser validate c:\myfiles\file.pp` in the Puppet command prompt. If a manifest has no syntax errors, the tool outputs nothing.
2. To see what output occurs when there is an error, temporarily edit the manifest and remove the : after the resource title. Run `puppet parser validate c:\myfiles\file.pp` again, and see the following output:

   ```
   Error: Could not parse for environment production: Syntax error at 'ensure' at c:/myfiles/file.pp:2:3
   ```

© 2022 Puppet, Inc., a Perforce company
Launch the Puppet command prompt
A lot of common interactions with Puppet are done via the command line.

To open the command line interface, enter Command Prompt Puppet in your Start Menu, and click Start
Command Prompt with Puppet.

The Puppet command prompt has a few details worth noting:

- Several important batch files live in the current working directory, C:\Program Files\Puppet Labs \Puppet\bin. The most important of these batch files is puppet.bat. Puppet is a Ruby based application, and puppet.bat is a wrapper around executing Puppet code through ruby.exe.
- Running the command prompt with Puppet rather than just the default Windows command prompt ensures that all of the Puppet tooling is in PATH, even if you change to a different directory.

Simulate a Puppet run with --noop
Puppet has a switch that you can use to test if manifests make the intended changes. This is referred to as non-enforcement or no-op mode.

To simulate changes, run puppet apply c:\myfiles\file.pp --noop in the command prompt:

```
C:\Program Files\Puppet Labs\Puppet\bin>puppet apply c:\myfiles\file.pp --noop
Notice: Compiled catalog for win-User.localdomain in environment production in 0.45 seconds
Notice: /Stage[main]/MainFile[C:\Temp\foo.txt]/ensure: current value absent, should be present (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Applied catalog in 0.03 seconds
```

Puppet shows you the changes it would make, but does not actually make the changes. It would create a new file at C:\Temp\foo.txt, but it hasn't, because you used --noop.

Enforce the desired state with puppet apply
When the output of the simulation shows the changes you intend to make, you can start enforcing these changes with the puppet apply command.

Run puppet apply c:\myfiles\file.pp.

To see more details about what this command did, you can specify additional options, such as --trace, --debug, or --verbose, which can help you diagnose problematic code. If puppet apply fails, Puppet outputs a full stack trace.

Puppet enforces the resource state you've described in file.pp, in this case guaranteeing that a file (c:\Temp\foo.txt) is present and has the contents This is some text in my file.

Understanding idempotency
A key feature of Puppet is its idempotency: the ability to repeatedly apply a manifest to guarantee a desired resource state on a system, with the same results every time.

If a given resource is already in the desired state, Puppet performs no actions. If a given resource is not in the desired state, Puppet takes whatever action is necessary to put the resource into the desired state. Idempotency enables Puppet to simulate resource changes without performing them, and lets you set up configuration management one time, fixing configuration drift without recreating resources from scratch each time Puppet runs.

To demonstrate how Puppet can be applied repeatedly to get the same results, change the manifest at c:\myfiles \file.pp to the following:

```
file { 'C:\Temp\foo.txt':
   ensure => present,
   content => 'I have changed my file content.'
}
```

© 2022 Puppet, Inc., a Perforce company
Apply the manifest by running `puppet apply c:\myfiles\file.pp`. Open `c:\Temp\foo.txt` and notice that Puppet changes the file’s contents.

Applying the manifest again with `puppet apply c:\myfiles\file.pp` results in no changes to the system, demonstrating that Puppet behaves idempotently.

Many of the samples in Puppet documentation assume that you have this basic understanding of creating and editing manifest files, and applying them with `puppet apply`.

**Additional command line tools**

Once you understand how to write manifests, validate them, and use `puppet apply` to enforce your changes, you’re ready to use commands such as `puppet agent`, `puppet resource`, and `puppet module install`.

**puppet agent**

Like `puppet apply`, the `puppet agent` command line tool applies configuration changes to a system. However, `puppet agent` retrieves compiled catalogs from a Puppet Server, and applies them to the local system. Puppet is installed as a Windows service, and by default tries to contact the primary server every 30 minutes by running `puppet agent` to retrieve new catalogs and apply them locally.

**puppet resource**

You can run `puppet resource` to query the state of a particular type of resource on the system. For example, to list all of the users on a system, run the command `puppet resource user`.

The computer used for this example has three local user accounts: Administrator, Guest, and vagrant. Note that the output is the same format as a manifest, and you can copy and paste it directly into a manifest.

**puppet module install**

Puppet includes many core resource types, plus you can extend Puppet by installing modules. Modules contain additional resource definitions and the code necessary to modify a system to create, read, modify, or delete those resources. The Puppet Forge contains modules developed by Puppet and community members available for anyone to use.

Puppet synchronizes modules from a primary server to agent nodes during `puppet agent` runs. Alternatively, you can use the standalone Puppet module tool, included when you install Puppet, to manage, view, and test modules.

Run `puppet module list` to show the list of modules installed on the system.
To install modules, the Puppet module tool uses the syntax `puppet module install NAMESPACE/MODULENAME`. The NAMESPACE is registered to a module, and MODULE refers to the specific module name. A very common module to install on Windows is registry, under the puppetlabs namespace. So, to install the registry module, run `puppet module install puppetlabs/registry`.

Manage Windows services

You can use Puppet to manage Windows services, specifically, to start, stop, enable, disable, list, query, and configure services. This way, you can ensure that certain services are always running or are disabled as necessary.

You write Puppet code to manage services in the manifest. When you apply the manifest, the changes you make to the service are applied.

**Note:** In addition to using manifests to apply configuration changes, you can query system state using the `puppet resource` command, which emits code as well as applying changes.

Ensure a Windows service is running

There are often services that you always want running in your infrastructure.

To have Puppet ensure that a service is running, use the following code:

```puppet
service { '<service name>':
  ensure => 'running'
}
```

**Example**

For example, the following manifest code ensures the Windows Time service is running:

```puppet
service { 'w32time':
  ensure => 'running'
}
```

Stop a Windows service

Some services can impair performance, or might need to be stopped for regular maintenance.

To disable a service, use the code:

```puppet
service { '<service name>':
  ensure => 'stopped',
  enable => 'false'
}
```

**Example**

For example, this disables the disk defragmentation service, which can negatively impact service performance.

```puppet
service { 'defragsvc':
  ensure => 'stopped',
  enable => 'false'
}
```

Schedule a recurring operation with Windows Task Scheduler

Regularly scheduled operations, or tasks, are often necessary on Windows to perform routine system maintenance.

**Note:** If you need to run an ad-hoc task in the PE console or on the command line, see Running tasks in PE on page 634.
If you need to sync files from another system on the network, perform backups to another disk, or execute log or index maintenance on SQL Server, you can use Puppet to schedule and perform regular tasks. The following shows how to regularly delete files.

To delete all files recursively from `C:\Windows\Temp` at 8 AM each day, create a resource called scheduled_task with these attributes:

```shell
scheduled_task { 'Purge global temp files':
  ensure => present,
  enabled => true,
  command => 'C:\Windows\system32\cmd.exe',
  arguments => '/c "del c:\Windows\temp\*.* /F /S /Q"',
  trigger => {
    schedule => daily,
    start_time => '08:00',
  }
}
```

After you set up Puppet to manage this task, the Task Scheduler includes the task you specified:

![Task Scheduler screenshot](image)

**Example**

In addition to creating a trivial daily task at a specified time, the scheduled task resource supports a number of other more advanced scheduling capabilities, including more fine-tuned scheduling. For example, to change the above task to instead perform a disk clean-up every 2 hours, modify the trigger definition:

```shell
scheduled_task { 'Purge global temp files every 2 hours':
  ensure => present,
  enabled => true,
  command => 'C:\Windows\system32\cmd.exe',
  arguments => '/c "del c:\Windows\temp\*.* /F /S /Q"',
}
```
trigger => [{
    day_of_week => ['mon', 'tues', 'wed', 'thurs', 'fri'],
    every => '1',
    minutes_interval => '120',
    minutes_duration => '1440',
    schedule => 'weekly',
    start_time => '07:30'
},
    user => 'system',
}]

You can see the corresponding definition reflected in the Task Scheduler GUI:

![Task Scheduler GUI](image)

Manage Windows users and groups

Puppet can be used to create local group and user accounts. Local user accounts are often desirable for isolating applications requiring unique permissions.

Manage administrator accounts

It is often necessary to standardize the local Windows Administrator password across an entire Windows deployment.

To manage administrator accounts with Puppet, create a user resource with 'Administrator' as the resource title like so:

```puppet
user { 'Administrator':
    ensure => present,
}
```
password => 'yabbadabba'
}

**Note:** Securing the password used in the manifest is beyond the scope of this introductory example, but it’s common to use Hiera, a key/value lookup tool for configuration, with eyaml to solve this problem. Not only does this solution provide secure storage for the password value, but it also provides parameterization to support reuse, opening the door to easy password rotation policies across an entire network of Windows machines.

### Configure an app to use a different account

You might not always want to use the default user for an application, you can use Puppet to create users for other applications, like ASP.NET.

To configure ASP.NET apps to use accounts other than the default Network Service, create a user and exec resource:

```yaml
user { 'aspnet_banking_app':
  ensure          => present,
  managehome      => true,
  comment         => 'ASP.NET Service account for Banking application',
  password        => 'banking_app_password',
  groups          => ['IIS_IUSRS', 'Users'],
  auth_membership => 'minimum',
  notify          => Exec['regiis_aspnet_banking_app']
}

exe { 'regiis_aspnet_banking_app':
  path        => 'c:\windows\Microsoft.NET\Framework\v4.0.30319',
  command     => 'aspnet_regiis.exe -ga aspnet_banking_app',
  refreshonly => true
}
```

In this example, the user is created in the appropriate groups, and the ASP.NET IIS registration command is run after the user is created to ensure file permissions are correct.
In the user resource, there are a few important details to note:

• **managehome** is set to create the user's home directory on disk.
• **auth_membership** is set to minimum, meaning that Puppet makes sure the **aspnet_banking_app** user is a part of the **IIS_IUSRS** and **Users** group, but doesn't remove the user from any other groups it might be a part of.
• **notify** is set on the user, and **refreshonly** is set on the exec. This tells Puppet to run **aspnet_regiis.exe** only when the **aspnet_banking_app** is created or changed.

**Manage local groups**
Local user accounts are often desirable for isolating applications requiring unique permissions. It can also be useful to manipulate existing local groups.

To add domain users or groups not present in the Domain Administrators group to the local Administrators group, use this code:

```powershell
-group { 'Administrators':
    ensure => 'present',
    members => ['DOMAIN\User'],
    auth_membership => false
}
```

In this case, **auth_membership** is set to false to ensure that DOMAIN\User is present in the Administrators group, but that other accounts that might be present in Administrators are not removed.

Note that the **groups** attribute of **user** and the **members** attribute of **group** might both accept SID values, like the well-known SID for Administrators, S-1-5-32-544.
Executing PowerShell code

Some Windows maintenance tasks require the use of Windows Management Instrumentation (WMI), and PowerShell is the most useful way to access WMI methods. Puppet has a special module that can be used to execute arbitrary PowerShell code.

A common Windows maintenance tasks is to disable Windows drive indexing, because it can negatively impact disk performance on servers.

To disable drive indexing:

```powershell
$drive = 'C:'
exec { 'disable-c-indexing':
    provider => powershell,
    command => "\$wmi_volume = Get-WmiObject -Class Win32_Volume -Filter 'DriveLetter="$\{drive\}\"'; if (\$wmi_volume.IndexingEnabled -ne \$True) {
        return
    }; \$wmi_volume | Set-WmiInstance -Arguments @{IndexingEnabled = \$False}",
    unless => "if ((Get-WmiObject -Class Win32_Volume -Filter 'DriveLetter="$\{drive\}\"').IndexingEnabled) { exit 1 }","}
```

You can see the results in your object editor window:

Using the Windows built-in WBEMTest tool, running this manifest sets IndexingEnabled to FALSE, which is the desired behavior.

This `exec` sets a few important attributes:

- The provider is configured to use PowerShell (which relies on the module).
• The command contains inline PowerShell, and as such, must be escaped with PowerShell variables preceded with `$` must be escaped as `\$`.
• The `unless` attribute is set to ensure that Puppet behaves idempotently, a key aspect of using Puppet to manage resources. If the resource is already in the desired state, Puppet does not modify the resource state.

**Using templates to better manage Puppet code**

While inline PowerShell is usable as an `exec` resource in your manifest, such code can be difficult to read and maintain, especially when it comes to handling escaping rules.

For executing multi-line scripts, use Puppet templates instead. The following example shows how you can use a template to organize the code for disabling Windows drive indexing.

```powershell
$drive = 'C:'
exec { 'disable-c-indexing':
    command => template('Disable-Indexing.ps1.erb'),
    provider => powershell,
    unless => "if ((Get-WmiObject -Class Win32_Volume -Filter 'DriveLetter=\"${drive}\"').IndexingEnabled) { exit 1 }",
}
```

The PowerShell code for `Disable-Indexing.ps1.erb` becomes:

```powershell
function Disable-Indexing($Drive)
{
    $drive = Get-WmiObject -Class Win32_Volume -Filter "DriveLetter='$Drive'"
    if ($drive.IndexingEnabled -ne $True) { return }
    $drive | Set-WmiInstance -Arguments @{IndexingEnabled=$False} | Out-Null
}
```

**Using Windows modules**

You can use modules from the Forge to perform basic management tasks on Windows nodes, such as managing access control lists and registry keys, and installing and creating your own packages.

**Manage permissions with the acl module**

The `puppetlabs-acl` module helps you manage access control lists (ACLs), which provide a way to interact with permissions for the Windows file system. This module enables you to set basic permissions up to very advanced permissions using SIDs (Security Identifiers) with an access mask, inheritance, and propagation strategies. First, start with querying some existing permissions.

**View file permissions with ACL**

ACL is a custom type and provider, so you can use `puppet resource` to look at existing file and folder permissions.

For some types, you can use the command `puppet resource <TYPE NAME>` to get all instances of that type. However, there could be thousands of ACLs on a Windows system, so it's best to specify the folder you want to review the types in. Here, check `c:\Users` to see what permissions it contains.

In the command prompt, enter `puppet resource acl c:\Users`

```powershell
acl { 'c:\Users':
    inherit_parent_permissions => 'false',
    permissions => [
        {identity => 'SYSTEM', rights => ['full']},
        {identity => 'Administrators', rights => ['full']},
        {identity => 'Users', rights => ['read', 'execute'], affects => 'self_only'],
    ],
```
As you can see, this particular folder does not inherit permissions from its parent folder; instead, it sets its own permissions and determines how child files and folders inherit the permissions set here.

- `{identity => 'SYSTEM', rights => ['full']}` states that the “SYSTEM” user has full rights to this folder, and by default all children and grandchildren files and folders (as these are the same defaults when creating permissions in Windows).
- `{identity => 'Users', rights => ['read', 'execute'], affects => 'self_only'}` gives read and execute permissions to Users but only on the current directory.
- `{identity => 'Everyone', rights => ['read', 'execute'], affects => 'children_only'}` gives read and execute permissions to everyone, but only on subfolders and files.

**Note:** You might see what appears to be the same permission for a user/group twice (both "Users" and "Everyone" above), where one affects only the folder itself and the other is about children only. They are in fact different permissions.

### Create a Puppet managed permission

1. Run this code to create your first Puppet managed permission. Then, save it as `perms.pp`

```puppet
data{'c:/tempperms':
  ensure => directory,
}
# By default, the acl creates an implicit relationship to any
# file resources it finds that match the location.
acl {'c:/tempperms':
  permissions => [
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute']}
  ],
}
```

2. To validate your manifest, in the command prompt, run `puppet parser validate c:\<FILE PATH>\perms.pp`. If the parser returns nothing, it means validation passed.

3. To apply the manifest, type `puppet apply c:\<FILE PATH>\perms.pp`.

Your output should look similar to:

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.12 seconds
Notice: /Stage[main]/Main/File[c:/tempperms]/ensure: created
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions changed [ ] to [ ]
{ identity => 'BUILTIN\Administrators', rights => ['full'] },
{ identity => 'BUILTIN\Users', rights => ['read', 'execute'] }]
Notice: Applied catalog in 0.05 seconds
```
4. Review the permissions in your Windows UI. In Windows Explorer, right-click `tempperms` and click Properties. Then, click the Security tab. It should appear similar to the image below.

5. Optional: It might appear that you have more permissions than you were hoping for here. This is because by default Windows inherits parent permissions. In this case, you might not want to do that. Adjust the acl resource to not inherit parent permissions by changing the `perms.pp` file to look like the below by adding `inherit_parent_permissions => false`.

```plaintext
cac {'c:/tempperms':
    inherit_parent_permissions => false,
    permissions => [
        {identity => 'Administrators', rights => ['full']},
        {identity => 'Users', rights => ['read','execute']}
    ],
},
```
6. Save the file, and return the command prompt to run `puppet parser validate c:\<FILE PATH>\perms.pp` again.

7. When it validates, run `puppet apply c:\<FILE PATH>\perms.pp`

You should get output similar to the following:

```
C:\>puppet apply c:\puppet_code\perms.pp
Notice: Compiled catalog for win2012r2x64 in environment production in 0.08 seconds
Notice: /Stage[main]/Main/Acl[c:/tempperms]/inherit_parent_permissions:
inherit_parent_permissions changed 'true' to 'false'
Notice: Applied catalog in 0.02 seconds
```

8. To check the permissions again, enter `icacls c:\tempperms` in the command prompt. The command, `icacls`, is specifically for displaying and modifying ACLs. The output should be similar to the following:

```
C:\>icacls c:\tempperms
c:\tempperms BUILTIN\Administrators:(OI)(CI)(F) BUILTIN\Users:(OI)(CI)(RX)
         NT AUTHORITY\SYSTEM:(OI)(CI)(F) BUILTIN\Users:(CI)(AD)
         CREATOR OWNER:(OI)(CI)(IO)(F)
Successfully processed 1 files; Failed processing 0 files
```

The output shows each permission, followed by a list of specific rights in parentheses. This output shows there are more permissions than you specified in `perms.pp`. Puppet manages permissions next to unmanaged or existing permissions. In the case of removing inheritance, by default Windows copies those existing inherited permissions (or Access Control Entries, ACEs) over to the existing ACL so you have some more permissions that you might not want.

9. Remove the extra permissions, so that only the permissions you’ve specified are on the folder. To do this, in your `perms.pp` set `purge => true` as follows:

```
acl {'c:/tempperms':
    inherit_parent_permissions => false,
    purge                    => true,
    permissions              => [
        {identity => 'Administrators', rights => ['full']},
        {identity => 'Users', rights => ['read','execute']}]
}
```

10. Run the parser command as you have before. If it still returns no errors, then you can apply the change.
11. To apply the change, run `puppet apply c:\<FILE PATH>\perms.pp`. The output should be similar to below:

```
C:\>puppet apply c:\puppet_code\perms.pp
Notice: Compiled catalog for win2012r2x64 in environment production in 0.08 seconds
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions changed [ { identity => 'BUILTIN\Administrators', rights => ["full"] }, { identity => 'BUILTIN\Users', rights => ["read", "execute"] }, { identity => 'NT AUTHORITY\SYSTEM', rights => ["full"] }, { identity => 'BUILTIN\Users', rights => ["mask_specific"], mask => '4', child_types => 'containers' }, { identity => 'CREATOR OWNER', rights => ["full"], affects => 'children_only' } ]
Notice: Applied catalog in 0.05 seconds
```

Puppet outputs a notice as it is removing each of the permissions.

12. Take a look at the output of `icacls` again with `icacls c:\tempperms`

```
c:\>icacls c:\tempperms

\rmpl: BUILTIN\Administrators:(OI)(CI)(F)
\rmpl: BUILTIN\Users:(OI)(CI)(RX)
Successfully processed 1 files; Failed processing 0 files
```

Now the permissions have been set up for this directory. You can get into more advanced permission scenarios if you read the usage scenarios on this module’s Forge page.

**Create managed registry keys with registry module**

You might eventually need to use the registry to access and set highly available settings, among other things. The `puppetlabs-registry` module, which is also a Puppet Supported Module enables you to set both registry keys and values.

**View registry keys and values with puppet resource**

`puppetlabs-registry` is a custom type and provider, so you can use `puppet resource` to look at existing registry settings.

It is also somewhat limited, like the `acl` module in that it is restricted to only what is specified.

1. In your command prompt, run: `puppet resource registry_key 'HKLM\Software\Microsoft\Windows'

```
C:\>puppet resource registry_key 'HKLM\Software\Microsoft\Windows' registry_key { 'HKLM\Software\Microsoft\Windows\':
  ensure => 'present',
}
```

Not that interesting, but now take a look at a registry value.
2. Enter puppet resource registry_value 'HKLM\SYSTEM\CurrentControlSet\Services\BITS\DisplayName'

```puppet
registry_value { 'HKLM\SYSTEM\CurrentControlSet\Services\BITS\DisplayName':
  ensure => 'present',
  data   => ['Background Intelligent Transfer Service'],
  type   => 'string',
}
```

That’s a bit more interesting than a registry key.

Keys are like file paths (directories) and values are like files that can have data and be of different types.

**Create managed keys**

Learn how to make managed registry keys, and see Puppet correct configuration drift when you try and alter them in Registry Editor.

1. Create your first Puppet managed registry keys and values:

```puppet
registry_key { 'HKLM\Software\zTemporaryPuppet':
  ensure => present,
}

# By default the registry creates an implicit relationship to any file resources it finds that match the location.
registry_value {'HKLM\Software\zTemporaryPuppet\StringValue':
  ensure => 'present',
  data   => 'This is a custom value.',
  type   => 'string',
}

# forcing a 32-bit registry view; watch where this is created:
registry_key { '32:HKLM\Software\zTemporaryPuppet':
  ensure => present,
}
registry_value {'32:HKLM\Software\zTemporaryPuppet\StringValue':
  ensure => 'present',
  data   => 'This is a custom 32-bit value.',
  type   => 'expand',
}
```

2. Save the file as registry.pp.

3. Validate the manifest. In the command prompt, run puppet parser validate c:\<FILE PATH>\registry.pp

If the parser returns nothing, it means validation passed.
4. Now, apply the manifest by running `puppet apply c:\<FILE PATH>\registry.pp` in the command prompt. Your output should look similar to below.

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.11 seconds
Notice: /Stage[main]/Main/Registry_key[HKLM\Software\zTemporaryPuppet]/ensure: created
Notice: /Stage[main]/Main/Registry_value[HKLM\Software\zTemporaryPuppet\StringValue]/ensure: created
Notice: /Stage[main]/Main/Registry_key[32:HKLM\Software\zTemporaryPuppet]/ensure: created
Notice: /Stage[main]/Main/Registry_value[32:HKLM\Software\zTemporaryPuppet\StringValue]/ensure: created
Notice: Applied catalog in 0.03 seconds
```

5. Next, inspect the registry and see what you have. Press `Start + R`, then type `regedit` and press `Enter`. Once the Registry Editor opens, find your keys under `HKEY_LOCAL_MACHINE`.

Note that the 32-bit keys were created under the 32-bit section of Wow6432Node for Software.

6. Apply the manifest again by running `puppet apply c:\<FILE PATH>\registry.pp`

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.11 seconds
Notice: Applied catalog in 0.02 seconds
```

Nothing changed, so there is no work for Puppet to do.
7. In Registry Editor, change the data. Select `HKLM\Software\zTemporaryPuppet` and in the right box, double-click `StringValue`. Edit the value data, and click OK.

![Registry Editor Screenshot]

This time, changes have been made, so running `puppet apply c:\path\to\registry.pp` results in a different output.

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.11 seconds
Notice: /Stage[main]/Main/Registry_value[HKLM\Software\zTemporaryPuppet \StringValue]/data: data changed 'This is a custom value. Edited' to 'This is a custom value.'
Notice: Applied catalog in 0.03 seconds
```

Puppet automatically corrects the configuration drift.

8. Next, clean up and remove the keys and values. Make your `registry.pp` file look like the below:

```puppet
registry_key { 'HKLM\Software\zTemporaryPuppet':
  ensure => absent,
}

# forcing a 32 bit registry view, watch where this is created
registry_key { '32:HKLM\Software\zTemporaryPuppet':
  ensure => absent,
}
```
9. Validate it with `puppet parser validate c:\path\to\registry.pp` and apply it again with `puppet apply c:\path\to\registry.pp`

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.06 seconds
Notice: /Stage[main]/Main/Registry_key[HKLM\Software\zTemporaryPuppet]/ensure: removed
Notice: /Stage[main]/Main/Registry_key[32:HKLM\Software\zTemporaryPuppet]/ensure: removed
Notice: Applied catalog in 0.02 seconds
```

Refresh the view in your Registry Editor. The values are gone.

**Example**

Here’s a real world example that disables error reporting:

```ruby
class puppetconf::disable_error_reporting {
  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\ForceQueue':
    type => dword,
    data => '1',
  }

  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\DontShowUI':
    type => dword,
    data => '1',
  }

  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\DontSendAdditionalData':
    type => dword,
    data => '1',
  }

  registry_key { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Consent':
    ensure => present,
  }
}
```
registry_value {  'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Consent\DefaultConsent':    type => 'dword',    data => '2',  }
}

Create, install, and repackage packages with the chocolatey module
Chocolatey is a package manager for Windows that is similar in design and execution to package managers on non-Windows systems. The chocolatey module is a Puppet Approved Module, so it's not eligible for Puppet Enterprise support services. The module has the capability to install and configure Chocolatey itself, and then manage software on Windows with Chocolatey packages.

View existing packages
Chocolatey has a custom provider for the package resource type, so you can use puppet resource to view existing packages.

In the command prompt, run puppet resource package --param provider | more

The additional provider parameter in this command outputs all types of installed packages that are detected by multiple providers.

Install Chocolatey
These steps are to install Chocolatey (choco.exe) itself. You use the module to ensure Chocolatey is installed.

1. Create a new manifest in the chocolatey module called chocolatey.pp with the following contents:
   include chocolatey

2. Validate the manifest by running puppet parser validate c:\<FILE PATH>\chocolatey.pp in the command prompt. If the parser returns nothing, it means validation passed.

3. Apply the manifest by running puppet apply c:\<FILE PATH>\chocolatey.pp

   Your output should look similar to below:

   Notice: Compiled catalog for win2012r2x64 in environment production in 0.58 seconds
   Notice: /Stage[main]/Chocolatey::Install/Windows_env[chocolatey_PATH_env]/ensure: created
   Notice: /Stage[main]/Chocolatey::Install/Windows_env[chocolatey_ChocolateyInstall_env]/ensure: created
   Notice: /Stage[main]/Chocolatey::Install/Exec[install_chocolatey_official]/return: executed successfully
   Notice: /Stage[main]/Chocolatey::Install/Exec[install_chocolatey_official]: Triggered 'refresh' from 2 events
   Notice: Finished catalog run in 13.22 seconds

In a production scenario, you’re likely to have a Chocolatey.nupkg file somewhere internal. In cases like that, you can use the internal nupkg for Chocolatey installation:

class {'chocolatey':
  chocolatey_download_url => 'https://internalurl/to/chocolatey.nupkg',
  use_7zip => false,
  log_output => true,
}

© 2022 Puppet, Inc., a Perforce company
Install a package with chocolatey

Normally, when installing packages you copy them locally first, make any required changes to bring everything they download to an internal location, repackage the package with the edits, and build your own packages to host on your internal package repository (feed). For this exercise, however, you directly install a portable Notepad++ from Chocolatey's community feed. The Notepad++ CommandLine package is portable and shouldn't greatly affect an existing system.

1. Update the manifest chocolatey.pp with the following contents:

```powershell
package {'notepadplusplus.commandline':
    ensure   => installed,
    provider => chocolatey,
}
```

2. Validate the manifest by running `puppet parser validate c:\<FILE PATH>\chocolatey.pp` in the command prompt. If the parser returns nothing, it means validation passed.

3. Now, apply the manifest with `puppet apply c:\<FILE PATH>\chocolatey.pp`. Your output should look similar to below.

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.75 seconds
Notice: /Stage[main]/Main/Package[notepadplusplus.commandline]/ensure: created
Notice: Applied catalog in 15.51 seconds
```

If you want to use this package for a production scenario, you need an internal custom feed. This is simple to set up with the chocolatey_server module. You could also use Sonatype Nexus, Artifactory, or a CIFS share if you want to host packages with a non-Windows option, or you can use anything on Windows that exposes a NuGet OData feed (Nuget is the packaging infrastructure that Chocolatey uses). See the How To Host Feed page of the chocolatey wiki for more in-depth information. You could also store packages on your primary server and use a file resource to verify they are in a specific local directory prior to ensuring the packages.

Example

The following example ensures that Chocolatey, the Chocolatey Simple Server (an internal Chocolatey package repository), and some packages are installed. It requires the additional chocolatey/chocolatey_server module.

In `c:\<FILE PATH>\packages` you must have packages for Chocolatey, Chocolatey.Server, RoundhousE, Launchy, and Git, as well as any of their dependencies for this to work.

```powershell
case $operatingsystem {
    'windows': {
        Package {
            provider => chocolatey,
            source   => 'C:\packages',
        }
    }
}

# include chocolatey
class {'chocolatey':
    chocolatey_download_url => 'file:///C:/packages/chocolatey.0.9.9.11.nupkg',
    use_7zip => false,
    log_output => true,
}

# This contains the bits to install the custom server.
# include chocolatey_server
class {'chocolatey_server':
    server_package_source => 'C:/packages',
}
```

© 2022 Puppet, Inc., a Perforce company
Copy an existing package and make it internal (repackaging packages)
To make the existing package local, use these steps.

Chocolety's community feed has quite a few packages, but they are geared towards community and use the internet for downloading from official distribution sites. However, they are attractive as they have everything necessary to install a piece of software on your machine. Through the repackaging process, by which you take a community package and bring all of the bits internal or embed them into the package, you can completely internalize a package to host on an internal Chocolatey/NuGet repository. This gives you complete control over a package and removes the aforementioned production trust and control issues.

1. Download the Notepad++ package from Chocolatey's community feed by going to the package page and clicking
   the download link.
2. Rename the downloaded file to end with .zip and unpack the file as a regular archive.
3. Delete the _rels and package folders and the [Content_Types].xml file. These are created during choco pack and should not be included, because they're regenerated (and their existence leads to issues).

4. Open tools\chocolateyInstall.ps1.

   ```powershell
   Install-ChocolateyZipPackage 'notepadplusplus.commandline' 'https://notepad-plus-plus.org/repository/6.x/6.8.7/npp.6.8.7.bin.zip' "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
   ```

5. Download the zip file and place it in the tools folder of the package.
6. Next, edit chocolateyInstall.ps1 to point to this embedded file instead of reaching out to the internet
   (if the size of the file is over 50MB, you might want to put it on a file share somewhere internally for better
   performance).

   ```powershell
   $toolsDir   = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
   Install-ChocolateyZipPackage 'notepadplusplus.commandline' "$toolsDir\npp.6.8.7.bin.zip" "$toolsDir"
   ```

   The double quotes allow for string interpolation (meaning variables get interpreted instead of taken literally).
7. Next, open the *.nuspec file to view its contents and make any necessary changes.

```xml
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>notepadplusplus.commandline</id>
    <version>6.8.7</version>
    <title>Notepad++ (Portable, CommandLine)</title>
    <authors>Don Ho</authors>
    <owners>Rob Reynolds</owners>
    <projectUrl>https://notepad-plus-plus.org/</projectUrl>
    <iconUrl>https://cdn.rawgit.com/ferventcoder/chocolatey-packages/02c21bebe5abb495a56747cbb9b4b5415c933fc0/icons/notepadplusplus.png</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Notepad++ is a free source code editor and Notepad replacement that supports several languages.</description>
    <summary>Notepad++ is a free (as in "free speech" and also as in "free beer") source code editor and Notepad replacement that supports several languages.</summary>
    <tags>notepad notepadplusplus notepad-plus-plus</tags>
  </metadata>
</package>
```

Some organizations change the version field to denote that this is an edited internal package, for example changing 6.8.7 to 6.8.7.20151202. For now, this is not necessary.

8. Now you can navigate via the command prompt to the folder with the .nuspec file (from a Windows machine unless you’ve installed Mono and built choco.exe from source) and use choco pack. You can also be more specific and run choco pack <FILE PATH>
otepadplusplus.commandline.nuspec. The output should be similar to below.

```
Attempting to build package from 'notepadplusplus.commandline.nuspec'.
Successfully created package 'notepadplusplus.commandline.6.8.7.nupkg'
```

Normally you test on a system to ensure that the package you just built is good prior to pushing the package (just the *.nupkg) to your internal repository. This can be done by using choco.exe on a test system to install (choco install notepadplusplus.commandline -source %cd% -change %cd% to $pwd in PowerShell.exe) and uninstall (choco uninstall notepadplusplus.commandline). Another method of testing is to run the manifest pointed to a local source folder, which is what you are going to do.

9. Create c:\packages and copy the resulting package file (notepadplusplus.commandline.6.8.7.nupkg) into it.

This won’t actually install on this system since you just installed the same version from Chocolatey’s community feed. So you need to remove the existing package first. To remove it, edit your chocolatey.pp to set the package to absent.

```
package {'notepadplusplus.commandline':
  ensure  => absent,
  provider => chocolatey,
}
```

10. Validate the manifest with puppet parser validate path\to\chocolatey.pp. Apply the manifest to ensure the change puppet apply c:\path\to\chocolatey.pp.

You can validate that the package has been removed by checking for it in the package install location or by using choco list -lo.

© 2022 Puppet, Inc., a Perforce company
11. Update the manifest (chocolatey.pp) to use the custom package.

```yaml
package {'notepadplusplus.commandline':
    ensure => latest,
    provider => chocolatey,
    source => 'c:\packages',
}
```

12. Validate the manifest with the parser and then apply it again. You can see Puppet creating the new install in the output.

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.79 seconds
Notice: /Stage[main]/Main/Package[notepadplusplus.commandline]/ensure: created
Notice: Applied catalog in 14.78 seconds
```

13. In an earlier step, you added a *.zip file to the package, so that you can inspect it and be sure the custom package was installed. Navigate to `C:\ProgramData\chocolatey\lib\notepadplusplus.commandline\tools` (if you have a default install location for Chocolatey) and see if you can find the *.zip file.

You can also validate the `chocolateyInstall.ps1` by opening and viewing it to see that it is the custom file you changed.

**Create a package with chocolatey**

Creating your own packages is, for some system administrators, surprisingly simple compared to other packaging standards.

Ensure you have at least Chocolatey CLI (choco.exe) version 0.9.9.11 or newer for this next part.

1. From the command prompt, enter `choco new -h` to see a help menu of what the available options are.
2. Next, use `choco new vagrant` to create a package named 'vagrant'. The output should be similar to the following:

```
Creating a new package specification at C:\temppackages\vagrant
Generating template to a file at 'C:\temppackages\vagrant\vagrant.nuspec'
Generating template to a file at 'C:\temppackages\vagrant\tools\chocolateyinstall.ps1'
Generating template to a file at 'C:\temppackages\vagrant\tools\chocolateyuninstall.ps1'
Generating template to a file at 'C:\temppackages\vagrant\tools\ReadMe.md'
Successfully generated vagrant package specification files at 'C:\temppackages\vagrant'
```

It comes with some files already templated for you to fill out (you can also create your own custom templates for later use).
3. Open `vagrant.nuspec`, and edit it to look like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2015/06/nuspec.xsd">
  <metadata>
    <id>vagrant</id>
    <title>Vagrant (Install)</title>
    <version>1.8.4</version>
    <authors>HashiCorp</authors>
    <owners>my company</owners>
    <description>Vagrant - Development environments made easy.</description>
  </metadata>
  <files>
    <file src="tools\**" target="tools" />
  </files>
</package>
```

Unless you are sharing with the world, you don’t need most of what is in the nuspec template file, so only required items are included above. Match the version of the package in this nuspec file to the version of the underlying software as closely as possible. In this example, Vagrant 1.8.4 is being packaged.

4. Open `chocolateyInstall.ps1` and edit it to look like the following:

```powershell
$ErrorActionPreference = 'Stop';
(packageName = 'vagrant',
 $toolsDir   = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
 $fileLocation = Join-Path $toolsDir 'vagrant_1.8.4.msi'
 $packageArgs = @{
    packageName   = $packageName
    fileType      = 'msi'
    file         = $fileLocation
    silentArgs    = "/qn /norestart"
    validExitCodes= @(0, 3010, 1641)
}"
Install-ChocolateyInstallPackage @packageArgs
```

**Note:** The above is `Install-ChocolateyINSTALLPackage`, not to be confused with `Install-ChocolateyPackage`. The names are very close to each other, however the latter also downloads software from a URI (URL, ftp, file) which is not necessary for this example.

5. Delete the `ReadMe.md` and `chocolateyUninstall.ps1` files. Download Vagrant and move it to the tools folder of the package.

**Note:** Normally if a package is over 100MB, it is recommended to move the software installer/archive to a share drive and point to it instead. For this example, just bundle it as is.

6. Now pack it up by using `choco pack`. Copy the new `vagrant.1.8.4.nupkg` file to `c:\packages`.

7. Open the manifest, and add the new package you just created. Your `chocolatey.pp` file should look like the below.

```yaml
package {'vagrant'}:
  ensure => installed,
  provider => chocolatey,
  source  => 'c:\packages',
```

© 2022 Puppet, Inc., a Perforce company
8. Save the file and make sure to validate with the Puppet parser.
9. Use `puppet apply <FILE PATH>\chocolatey.pp` to run the manifest.
10. Open **Control Panel, Programs and Features** and take a look.

![Uninstall or change a program](image)

Vagrant is installed!

**Uninstall packages with Chocolatey**

In addition to installing and creating packages, Chocolatey can also help you uninstall them.

To verify that the choco `autoUninstaller` feature is turned on, use `choco feature` to list the features and their current state. If you're using `include chocolatey` or `class chocolatey` to ensure Chocolatey is installed, the configuration is applied automatically (unless you have explicitly disabled it). Starting in Chocolatey version 0.9.10, it is enabled by default.

1. If you see `autoUninstaller - [Disabled]`, you need to enable it. To do this, in the command prompt, run `choco feature enable -n autoUninstaller` You should see a similar success message:

   ```shell
   Enabled autoUninstaller
   ```

2. To remove Vagrant, edit your `chocolatey.pp` manifest to ensure => absent. Then save and validate the file.

   ```ruby
   package {'vagrant':
     ensure => absent,
     provider => chocolatey,
     source => 'c:\packages',
   }
   ```

3. Next, run `puppet apply <FILE PATH>\chocolatey.pp` to apply the manifest.

   ```bash
   Notice: Compiled catalog for win2012r2x64 in environment production in 0.75 seconds
   Notice: /Stage[main]/Main/Package[vagrant]/ensure: removed
   Notice: Applied catalog in 40.85 seconds
   ```

   You can look in the Control Panel, Programs and Features to see that it’s no longer installed!
Designing system configs: roles and profiles

Your typical goal with Puppet is to build complete system configurations, which manage all of the software, services, and configuration that you care about on a given system. The roles and profiles method can help keep complexity under control and make your code more reusable, reconfigurable, and refactorable.

- **The roles and profiles method** on page 502
  The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.
- **Roles and profiles example** on page 505
  This example demonstrates a complete roles and profiles workflow. Use it to understand the roles and profiles method as a whole. Additional examples show how to design advanced configurations by refactoring this example code to a higher level of complexity.
- **Designing advanced profiles** on page 508
  In this advanced example, we iteratively refactor our basic roles and profiles example to handle real-world concerns. The final result is — with only minor differences — the Jenkins profile we use in production here at Puppet.
- **Designing convenient roles** on page 524
  There are several approaches to building roles, and you must decide which ones are most convenient for you and your team.

**The roles and profiles method**

The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.

It's not a straightforward recipe; you must think hard about the nature of your infrastructure and your team. It's also not a final state; expect to refine your configurations over time. Instead, it's an approach to *designing your infrastructure's interface* — sealing away incidental complexity, surfacing the significant complexity, and making sure your data behaves predictably.
Building configurations without roles and profiles

Without roles and profiles, people typically build system configurations in their node classifier or main manifest, using Hiera to handle tricky inheritance problems. A standard approach is to create a group of similar nodes and assign classes to it, then create child groups with extra classes for nodes that have additional needs. Another common pattern is to put everything in Hiera, using a very large hierarchy that reflects every variation in the infrastructure.

If this works for you, then it works! You might not need roles and profiles. But most people find direct building gets difficult to understand and maintain over time.

Configuring roles and profiles

Roles and profiles are two extra layers of indirection between your node classifier and your component modules.

The roles and profiles method separates your code into three levels:

- Component modules — Normal modules that manage one particular technology, for example puppetlabs/apache.
- Profiles — Wrapper classes that use multiple component modules to configure a layered technology stack.
- Roles — Wrapper classes that use multiple profiles to build a complete system configuration.

These extra layers of indirection might seem like they add complexity, but they give you a space to build practical, business-specific interfaces to the configuration you care most about. A better interface makes hierarchical data easier to use, makes system configurations easier to read, and makes refactoring easier.

In short, from top to bottom:

- Your node classifier assigns one role class to a group of nodes. The role manages a whole system configuration, so no other classes are needed. The node classifier does not configure the role in any way.
- That role class declares some profile classes with include, and does nothing else. For example:

```rubular
class role::jenkins::master {
    include profile::base
}
```
Each profile configures a layered technology stack, using multiple component modules and the built-in resource types. (In the diagram, profile::jenkins::master uses puppet/jenkins, puppetlabs/apt, a home-built backup module, and some package and file resources.)

Profiles can take configuration data from the console, Hiera, or Puppet lookup. (In the diagram, three different hierarchy levels contribute data.)

Classes from component modules are always declared via a profile, and never assigned directly to a node.

- If a component class has parameters, you specify them in the profile; never use Hiera or Puppet lookup to override component class params.

Rules for profile classes
There are rules for writing profile classes.

- Make sure you can safely include any profile multiple times — don't use resource-like declarations on them.
- Profiles can include other profiles.
- Profiles own all the class parameters for their component classes. If the profile omits one, that means you definitely want the default value; the component class shouldn't use a value from Hiera data. If you need to set a class parameter that was omitted previously, refactor the profile.
- There are three ways a profile can get the information it needs to configure component classes:
  - If your business always uses the same value for a given parameter, hardcode it.
  - If you can't hardcode it, try to compute it based on information you already have.
  - Finally, if you can't compute it, look it up in your data. To reduce lookups, identify cases where multiple parameters can be derived from the answer to a single question.

This is a game of trade-offs. Hardcoded parameters are the easiest to read, and also the least flexible. Putting values in your Hiera data is very flexible, but can be very difficult to read: you might have to look through a lot of files (or run a lot of lookup commands) to see what the profile is actually doing. Using conditional logic to derive a value is a middle-ground. Aim for the most readable option you can get away with.

Rules for role classes
There are rules for writing role classes.

- The only thing roles should do is declare profile classes with include. Don't declare any component classes or normal resources in a role.

  Optionally, roles can use conditional logic to decide which profiles to use.
- Roles should not have any class parameters of their own.
- Roles should not set class parameters for any profiles. (Those are all handled by data lookup.)
- The name of a role should be based on your business's conversational name for the type of node it manages.

  This means that if you regularly call a machine a "Jenkins master," it makes sense to write a role named role::jenkins::master. But if you call it a "web server," you shouldn't use a name like role::nginx — go with something like role::web instead.

Methods for data lookup
Profiles usually require some amount of configuration, and they must use data lookup to get it.

This profile uses the automatic class parameter lookup to request data.

```bash
include profile::server
include profile::jenkins::master
}
```

# Example Hiera data
profile::jenkins::jenkins_port: 8000
profile::jenkins::java_dist: jre
profile::jenkins::java_version: '8'

# Example manifest
This profile omits the parameters and uses the `lookup` function:

```ruby
class profile::jenkins {
  Integer $jenkins_port,
  String $java_dist,
  String $java_version
}
#
```

In general, class parameters are preferable to lookups. They integrate better with tools like Puppet strings, and they're a reliable and well-known place to look for configuration. But using `lookup` is a fine approach if you aren't comfortable with automatic parameter lookup. Some people prefer the full lookup key to be written in the profile, so they can globally grep for it.

**Roles and profiles example**

This example demonstrates a complete roles and profiles workflow. Use it to understand the roles and profiles method as a whole. Additional examples show how to design advanced configurations by refactoring this example code to a higher level of complexity.

**Configure Jenkins master servers with roles and profiles**

Jenkins is a continuous integration (CI) application that runs on the JVM. The Jenkins master server provides a web front-end, and also runs CI tasks at scheduled times or in reaction to events.

In this example, we manage the configuration of Jenkins master servers.

**Set up your prerequisites**

If you're new to using roles and profiles, do some additional setup before writing any new code.

1. Create two modules: one named `role`, and one named `profile`.
   - If you deploy your code with Code Manager or r10k, put these two modules in your control repository instead of declaring them in your Puppetfile, because Code Manager and r10k reserve the `modules` directory for their own use.
     a. Make a new directory in the repo named `site`.
     b. Edit the `environment.conf` file to add `site` to the `modulepath`. (For example: `modulepath = site:modules:$basemodulepath`).
     c. Put the `role` and `profile` modules in the `site` directory.
2. Make sure Hiera or Puppet lookup is set up and working, with a hierarchy that works well for you.

**Choose component modules**

For our example, we want to manage Jenkins itself using the `puppet/jenkins` module.

Jenkins requires Java, and the `puppet/jenkins` module can manage it automatically. But we want finer control over Java, so we're going to disable that. So, we need a Java module, and `puppetlabs/java` is a good choice.

That's enough to start with. We can refactor and expand when we have those working.

To learn more about these modules, see `puppet/jenkins` and `puppetlabs/java`.  

© 2022 Puppet, Inc., a Perforce company
Write a profile

From a Puppet perspective, a profile is just a normal class stored in the profile module.

Make a new class called `profile::jenkins::master` located at `/profile/manifests/jenkins/master.pp`, and fill it with Puppet code.

```puppet
# /etc/puppetlabs/code/environments/production/site/profile/manifests/jenkins/master.pp
class profile::jenkins::master(
    String $jenkins_port = '9091',
    String $java_dist    = 'jdk',
    String $java_version = 'latest',
) {
    class { 'jenkins':
        configure_firewall => true,
        install_java       => false,
        port               => $jenkins_port,
        config_hash        => {
            'HTTP_PORT'    => { 'value' => $jenkins_port },
            'JENKINS_PORT' => { 'value' => $jenkins_port },
        },
    }
    class { 'java':
        distribution => $java_dist,
        version      => $java_version,
        before       => Class['jenkins'],
    }
}
```

This is pretty simple, but is already benefiting us: our interface for configuring Jenkins has gone from 30 or so parameters on the Jenkins class (and many more on the Java class) down to three. Notice that we’ve hardcoded the `configure_firewall` and `install_java` parameters, and have reused the value of `$jenkins_port` in three places.

Related information

Rules for profile classes on page 504

There are rules for writing profile classes.

Methods for data lookup on page 504

Profiles usually require some amount of configuration, and they must use data lookup to get it.

Set data for the profile

Let’s assume the following:

- We use some custom facts:
  - `group`: The group this node belongs to. (This is usually either a department of our business, or a large-scale function shared by many nodes.)
  - `stage`: The deployment stage of this node (dev, test, or prod).
- We have a five-layer hierarchy:
  - `console_data` for data defined in the console.
  - `nodes/%{trusted.certname}` for per-node overrides.
  - `groups/%{facts.group}/%{facts.stage}` for setting stage-specific data within a group.
  - `groups/%{facts.group}` for setting group-specific data.
  - `common` for global fallback data.
- We have a few one-off Jenkins masters, but most of them belong to the `ci` group.
• Our quality engineering department wants masters in the `ci` group to use the Oracle JDK, but one-off machines can just use the platform’s default Java.
• QE also wants their prod masters to listen on port 80.

Set appropriate values in the data, using either Hiera or configuration data in the console.

```yaml
# /etc/puppetlabs/code/environments/production/data/nodes/ci-master01.example.com.yaml
# --Nothing. We don't need any per-node values right now.

# /etc/puppetlabs/code/environments/production/data/groups/ci/prod.yaml
profile::jenkins::master::jenkins_port: '80'

# /etc/puppetlabs/code/environments/production/data/groups/ci.yaml
profile::jenkins::master::java_dist: 'oracle-jdk8'
profile::jenkins::master::java_version: '8u92'

# /etc/puppetlabs/code/environments/production/data/common.yaml
# --Nothing. Just use the default parameter values.
```

Write a role
To write roles, we consider the machines we’ll be managing and decide what else they need in addition to that Jenkins profile.

Our Jenkins masters don’t serve any other purpose. But we have some profiles (code not shown) that we expect every machine in our fleet to have:

- `profile::base` must be assigned to every machine, including workstations. It manages basic policies, and uses some conditional logic to include OS-specific profiles as needed.
- `profile::server` must be assigned to every machine that provides a service over the network. It makes sure ops can log into the machine, and configures things like timekeeping, firewalls, logging, and monitoring.

So a role to manage one of our Jenkins masters should include those classes as well.

```plaintext
class role::jenkins::master {
    include profile::base
    include profile::server
    include profile::jenkins::master
}
```

Related information
Rules for role classes on page 504
There are rules for writing role classes.

Assign the role to nodes
Finally, we assign `role::jenkins::master` to every node that acts as a Jenkins master.

Puppet has several ways to assign classes to nodes, so use whichever tool you feel best fits your team. Your main choices are:

- The console node classifier, which lets you group nodes based on their facts and assign classes to those groups.
- The main manifest which can use node statements or conditional logic to assign classes.
- Hiera or Puppet lookup — Use the `lookup` function to do a unique array merge on a special `classes` key, and pass the resulting array to the `include` function.

```plaintext
# /etc/puppetlabs/code/environments/production/manifests/site.pp
lookup('classes', {merge => unique}).include
```

To learn more about how to assign custom roles to individual nodes, visit https://puppet.com/docs/puppet/6/fact_overview.html

© 2022 Puppet, Inc., a Perforce company
Designing advanced profiles

In this advanced example, we iteratively refactor our basic roles and profiles example to handle real-world concerns. The final result is — with only minor differences — the Jenkins profile we use in production here at Puppet.

Along the way, we explain our choices and point out some of the common trade-offs you encounter as you design your own profiles.

Here's the basic Jenkins profile we're starting with:

```puppet
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master {  
  String $jenkins_port = '9091',  
  String $java_dist = 'jdk',  
  String $java_version = 'latest',  
}  

class { 'jenkins':  
  configure_firewall => true,  
  install_java => false,  
  port => $jenkins_port,  
  config_hash => {  
    'HTTP_PORT' => { 'value' => $jenkins_port },  
    'JENKINS_PORT' => { 'value' => $jenkins_port },  
  },  
}  

class { 'java':  
  distribution => $java_dist,  
  version => $java_version,  
  before => Class['jenkins'],  
}
```

Related information

Rules for profile classes on page 504
There are rules for writing profile classes.

First refactoring: Split out Java

We want to manage Jenkins masters and Jenkins agent nodes. We won't cover agent profiles in detail, but the first issue we encountered is that they also need Java.

We could copy and paste the Java class declaration; it's small, so keeping multiple copies up-to-date might not be too burdensome. But instead, we decided to break Java out into a separate profile. This way we can manage it one time, then include the Java profile in both the agent and master profiles.

Note: This is a common trade-off. Keeping a chunk of code in only one place (often called the DRY — "don't repeat yourself" — principle) makes it more maintainable and less vulnerable to rot. But it has a cost: your individual profile classes become less readable, and you must view more files to see what a profile actually does. To reduce that readability cost, try to break code out in units that make inherent sense. In this case, the Java profile's job is simple enough to guess by its name — your colleagues don't have to read its code to know that it manages Java 8. Comments can also help.

First, decide how configurable Java needs to be on Jenkins machines. After looking at our past usage, we realized that we use only two options: either we install Oracle's Java 8 distribution, or we default to OpenJDK 7, which the Jenkins module manages. This means we can:

- Make our new Java profile really simple: hardcode Java 8 and take no configuration.
- Replace the two Java parameters from profile::jenkins::master with one Boolean parameter (whether to let Jenkins handle Java).
Note: This is rule 4 in action. We reduce our profile's configuration surface by combining multiple questions into one.

Here's the new parameter list:

```puppet
class profile::jenkins::master {
  String $jenkins_port = '9091',
  Boolean $install_jenkins_java = true,
}
```

And here's how we choose which Java to use:

```puppet
class { 'jenkins':
  configure_firewall => true,
  install_java => $install_jenkins_java,    # <--- here
  port => $jenkins_port,
  config_hash => {
    'HTTP_PORT' => { 'value' => $jenkins_port },
    'JENKINS_PORT' => { 'value' => $jenkins_port },
  },
}
```

# When not using the jenkins module's java version, install java8.
unless $install_jenkins_java { include profile::jenkins::usage::java8 }

And our new Java profile:

```puppet
::jenkins::usage::java8
# Sets up java8 for Jenkins on Debian
#
class profile::jenkins::usage::java8 {
  motd::register { 'Java usage profile (profile::jenkins::usage::java8)': }
  # OpenJDK 7 is already managed by the Jenkins module.
  # ::jenkins::install_java or ::jenkins::agent::install_java should be false to use this profile
  # this can be set through the class parameter $install_jenkins_java
  case $::osfamily {
    'debian': {
      class { 'java':
        distribution => 'oracle-jdk8',
        version => '8u92',
      }
      package { 'tzdata-java':
        ensure => latest,
      }
    }
    default: {
      notify { "profile::jenkins::usage::java8 cannot set up JDK on ${::osfamily}": }
    }
  }
}
```

Diff of first refactor

```diff
@@ -1,13 +1,12 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/jenkins/master.pp
class profile::jenkins::master {
-  String $jenkins_port = '9091',
-  String $java_dist = 'jdk',
```

© 2022 Puppet, Inc., a Perforce company
Second refactor: Manage the heap

At Puppet, we manage the Java heap size for the Jenkins app. Production servers didn't have enough memory for heavy use.

The Jenkins module has a `jenkins::sysconfig` defined type for managing system properties, so let's use it:

```
# Manage the heap size on the master, in MB.
if($::memorysize_mb =~ Number and $::memorysize_mb > 8192)
{
    # anything over 8GB we should keep max 4GB for OS and others
    $heap = sprintf('%0.f', $::memorysize_mb - 4096)
} else {
    # This is calculated as 50% of the total memory.
    $heap = sprintf('%0.f', $::memorysize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
    value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled
}
```

**Note:** Rule 4 again — we couldn't hardcode this, because we have some smaller Jenkins masters that can't spare the extra memory. But because our production masters are always on more powerful machines, we can calculate the heap based on the machine's memory size, which we can access as a fact. This lets us avoid extra configuration.

Diff of second refactor

```
@@ -16,4 +16,20 @@ class profile::jenkins::master {
    # When not using the jenkins module's java version, install java8.
    unless $install_jenkins_java { include profile::jenkins::usage::java8 }
} +
```
### Third refactor: Pin the version
We dislike surprise upgrades, so we pin Jenkins to a specific version. We do this with a direct package URL instead of by adding Jenkins to our internal package repositories. Your organization might choose to do it differently.

First, we add a parameter to control upgrades. Now we can set a new value in .../data/groups/ci/dev.yaml while leaving .../data/groups/ci.yaml alone — our dev machines get the new Jenkins version first, and we can ensure everything works as expected before upgrading our prod machines.

```profile::jenkins::master {
    Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
    # ...
} (# ...)```

Then, we set the necessary parameters in the Jenkins class:

```class { 'jenkins':
    lts => true,              # <-- here
    repo => true,             # <-- here
    direct_download => $direct_download,  # <-- here
    version => 'latest',     # <-- here
    service_enable => true,  # <-- here
    service_ensure => running,
    configure_firewall => true,
    install_java => $install_jenkins_java,
    port => $jenkins_port,
    config_hash => {
        'HTTP_PORT' => { 'value' => $jenkins_port },
        'JENKINS_PORT' => { 'value' => $jenkins_port },
    },
} ```

This was a good time to explicitly manage the Jenkins service, so we did that as well.

### Diff of third refactor
@@ -1,10 +1,17 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifets/
jenkins/master.pp
class profile::jenkins::master {
```
Fourth refactor: Manually manage the user account
We manage a lot of user accounts in our infrastructure, so we handle them in a unified way. The `profile::server` class pulls in `virtual::users`, which has a lot of virtual resources we can selectively realize depending on who needs to log into a given machine.

Note: This has a cost — it's action at a distance, and you need to read more files to see which users are enabled for a given profile. But we decided the benefit was worth it: because all user accounts are written in one or two files, it's easy to see all the users that might exist, and ensure that they're managed consistently.

We're accepting difficulty in one place (where we can comfortably handle it) to banish difficulty in another place (where we worry it would get out of hand). Making this choice required that we know our colleagues and their comfort zones, and that we know the limitations of our existing code base and supporting services.

So, for this example, we change the Jenkins profile to work the same way; we manage the `jenkins` user alongside the rest of our user accounts. While we're doing that, we also manage a few directories that can be problematic depending on how Jenkins is packaged.

Some values we need are used by Jenkins agents as well as masters, so we're going to store them in a params class, which is a class that sets shared variables and manages no resources. This is a heavyweight solution, so wait until it provides real value before using it. In our case, we had a lot of OS-specific agent profiles (not shown in these examples), and they made a params class worthwhile.

Note: Just as before, "don't repeat yourself" is in tension with "keep it readable." Find the balance that works for you.
Three things to notice in the code above:

- We manage users with a homegrown `account::user` defined type, which declares a `user` resource plus a few other things.
- We use an `Account::User` resource collector to realize the Jenkins user. This relies on `profile::server` being declared.
- We set the Jenkins class's `manage_user`, `manage_group`, and `manage_datadirs` parameters to false.
- We're now explicitly managing the `plugins` directory and the `run` directory.

### Diff of fourth refactor

```plaintext
@@ -5,6 +5,33 @@ class profile::jenkins::master (
     Boolean $install_jenkins_java = true,
 ) {
+    # We rely on virtual resources that are ultimately declared by
+    # profile::server.
+    include profile::server
+    +
+    +    # Some default values that vary by OS:
+    +    include profile::jenkins::params
+    +    $jenkins_owner = $profile::jenkins::params::jenkins_owner
+    +    $jenkins_group = $profile::jenkins::params::jenkins_group
+    +    $master_config_dir = $profile::jenkins::params::master_config_dir
+    +
+    file { '/var/run/jenkins': ensure => 'directory' }
+    +
+    +    # Because our account::user class manages the '${master_config_dir}'
+    +    # directory
+    +    # as the 'jenkins' user's homedir (as it should), we need to manage
+    +    # '${master_config_dir}/plugins' here to prevent the upstream
```
pe | Managing nodes | 514

+ # rtyler-jenkins module from trying to manage the homedir as the config
+ # dir. For more info, see the upstream module's `manifests/plugin.pp`
+ # manifest.
+ file { "${master_config_dir}/plugins":
    ensure => directory,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0755',
    require => [Group[$jenkins_group], User[$jenkins_owner]],
+ }
+ Account::User <| tag == 'jenkins' |> +
+ class { 'jenkins':
    lts                => true,
    repo               => true,
@@ -14,6 +41,9 @@ class profile::jenkins::master (  service_ensure     => running,
      configure_firewall => true,
      install_java      => $install_jenkins_java,
+    manage_user        => false,
+    manage_group       => false,
+    manage_datadirs    => false,
      port               => $jenkins_port,
      config_hash        => {
        'HTTP_PORT'    => { 'value' => $jenkins_port },
    Fifth refactor: Manage more dependencies
Jenkins always needs Git installed (because we use Git for source control at Puppet), and it needs SSH keys to access private Git repos and run commands on Jenkins agent nodes. We also have a standard list of Jenkins plugins we use, so we manage those too.

Managing Git is pretty easy:

```plaintext
package { 'git':
    ensure => present,
}
```

SSH keys are less easy, because they are sensitive content. We can't check them into version control with the rest of our Puppet code, so we put them in a custom mount point on one specific Puppet server.

Because this server is different from our normal Puppet servers, we made a rule about accessing it: you must look up the hostname from data instead of hardcoding it. This lets us change it in only one place if the secure server ever moves.

```plaintext
$secure_server = lookup('puppetlabs::ssl::secure_server')

file { "${master_config_dir}/.ssh":
    ensure => directory,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0700',
}

file { "${master_config_dir}/.ssh/id_rsa":
    ensure => file,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0600',
    source => "puppet://$secure_server/=secure/delivery/id_rsa-jenkins",
}
```
Plugins are also a bit tricky, because we have a few Jenkins masters where we want to manually configure plugins. So we put the base list in a separate profile, and use a parameter to control whether we use it.

```puppet
class profile::jenkins::master ($manage_plugins = false, # ...
) {
  # ...
  if $manage_plugins {
    include profile::jenkins::master::plugins
  }
}
```

In the plugins profile, we can use the `jenkins::plugin` resource type provided by the Jenkins module.

```puppet
class profile::jenkins::master::plugins {
  jenkins::plugin { 'audit2db':          }
  jenkins::plugin { 'credentials':       }
  jenkins::plugin { 'jquery':            }
  jenkins::plugin { 'job-import-plugin': }
  jenkins::plugin { 'ldap':              }
  jenkins::plugin { 'mailer':            }
  jenkins::plugin { 'metadata':          }
  # ... and so on.
}
```

**Diff of fifth refactor**

```diff
@@ -1,6 +1,7 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master ($manage_plugins = false,
+  $jenkins_port = '9091',
  $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
  $install_jenkins_java = true,
) {
  $jenkins_group          = $profile::jenkins::params::jenkins_group
  $master_config_dir      = $profile::jenkins::params::master_config_dir
  if $manage_plugins {
    # About 40 jenkins::plugin resources:
    + include profile::jenkins::master::plugins
    + }
    + # Sensitive info (like SSH keys) isn't checked into version control like the
    + # rest of our modules; instead, it's served from a custom mount point on a
```

© 2022 Puppet, Inc., a Perforce company
+ # designated server.
+ $secure_server = lookup('puppetlabs::ssl::secure_server')
+
+ package { 'git':
+   ensure => present,
+ }
+
+ file { '/var/run/jenkins': ensure => 'directory' }

    # Because our account::user class manages the '${master_config_dir}'
    directory
    @@ -69,4 +84,29 @@ class profile::jenkins::master (  
              value => '-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
    }  
+
+ # Deploy the SSH keys that Jenkins needs to manage its agent machines and
+ # access Git repos.
+
+ file { '${master_config_dir}/.ssh':
+  ensure => directory,
+  owner => $jenkins_owner,
+  group => $jenkins_group,
+  mode => '0700',
+ }  
+
+ file { '${master_config_dir}/.ssh/id_rsa':
+  ensure => file,
+  owner => $jenkins_owner,
+  group => $jenkins_group,
+  mode => '0600',
+  source => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins",
+ }  
+
+ file { '${master_config_dir}/.ssh/id_rsa.pub':
+  ensure => file,
+  owner => $jenkins_owner,
+  group => $jenkins_group,
+  mode => '0640',
+  source => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins.pub",
+ }  
+
}

Sixth refactor: Manage logging and backups
Back up is usually a good idea.

We can use our homegrown backup module, which provides a backup:::job resource type
(profile::server takes care of its prerequisites). But we should make backups optional, so people don't
accidentally post junk to our backup server if they're setting up an ephemeral Jenkins instance to test something.

class profile::jenkins::master (  
    Boolean $backups_enabled = false,
    # ...
) {  
    # ...
    if $backups_enabled {  
      backup::job { "jenkins-data-${::hostname}"
          files => $master_config_dir,
      }
    }
  }
Also, our teams gave us some conflicting requests for Jenkins logs:

- Some people want it to use syslog, like most other services.
- Others want a distinct log file so syslog doesn't get spammed, and they want the file to rotate more quickly than it does by default.

That implies a new parameter. We can make one called `$jenkins_logs_to_syslog` and default it to `undef`. If you set it to a standard syslog facility (like `daemon.info`), Jenkins logs there instead of its own file.

We use `jenkins::sysconfig` and our homegrown `logrotate::job` to do the work:

```erb
class profile::jenkins::master (
  Optional[String[1]] $jenkins_logs_to_syslog = undef,
  # ...
) {
  # ...
  if $jenkins_logs_to_syslog {
    jenkins::sysconfig { 'JENKINS_LOG':
      value => "$jenkins_logs_to_syslog",
    }
  }
  # ...
  logrotate::job { 'jenkins':
    log => '/var/log/jenkins/jenkins.log',
    options => [
      'daily',
      'copytruncate',
      'missingok',
      'rotate ?',
      'compress',
      'delaycompress',
      'notifempty'
    ],
  }
}
```

**Diff of sixth refactor**

```diff
@@ -1,8 +1,10 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master ( String $jenkins_port = '9091',
  + Boolean $backups_enabled = false,
  Boolean $manage_plugins = false,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
+  Optional[String[1]] $jenkins_logs_to_syslog = undef,
  Boolean $install_jenkins_java = true,
) {
@@ -84,6 +86,15 @@ class profile::jenkins::master ( value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
  }
+  # Forward jenkins master logs to syslog.
+  # When set to facility.level the jenkins_log uses that value instead of a
```
Seventh refactor: Use a reverse proxy for HTTPS

We want the Jenkins web interface to use HTTPS, which we can accomplish with an Nginx reverse proxy. We also want to standardize the ports: the Jenkins app always binds to its default port, and the proxy always serves over 443 for HTTPS and 80 for HTTP.

If we want to keep vanilla HTTP available, we can provide an $ssl parameter. If set to false (the default), you can access Jenkins via both HTTP and HTTPS. We can also add a $site_alias parameter, so the proxy can listen on a hostname other than the node’s main FQDN.

```puppet
class profile::jenkins::master (
  Boolean                     $ssl = false,
  Optional[String[1]]         $site_alias = undef,
  # IMPORTANT: notice that $jenkins_port is removed.
  # ...
)

Set configure_firewall => false in the Jenkins class:

```puppet
class { 'jenkins':
  lts                => true,
  repo                => true,
}
We need to deploy SSL certificates where Nginx can reach them. Because we serve a lot of things over HTTPS, we already had a profile for that:

```bash
# Deploy the SSL certificate/chain/key for sites on this domain.
include profile::ssl::delivery_wildcard
```

This is also a good time to add some info for the message of the day, handled by puppetlabs/motd:

```bash
motd::register { 
  'Jenkins CI master (profile::jenkins::master)'
}
if $site_alias {
  motd::register { 
    'jenkins-site-alias':
    content => @("END"),
    profile::jenkins::master::proxy
    Jenkins site alias: ${site_alias}
  }-END
  order => 25,
}
```

The bulk of the work is handled by a new profile called `profile::jenkins::master::proxy`. We're omitting the code for brevity; in summary, what it does is:

- Include `profile::nginx`.
- Use resource types from the jfryman/nginx to set up a vhost, and to force a redirect to HTTPS if we haven't enabled vanilla HTTP.
- Set up logstash forwarding for access and error logs.
- Include `profile::fw::https` to manage firewall rules, if necessary.

Then, we declare that profile in our main profile:

```bash
class { 'profile::jenkins::master::proxy':
  site_alias  => $site_alias,
  require_ssl => $ssl,
}
```

**Important:**

We are now breaking rule 1, the most important rule of the roles and profiles method. Why?

Because `profile::jenkins::master::proxy` is a "private" profile that belongs solely to `profile::jenkins::master`. It will never be declared by any role or any other profile.

This is the only exception to rule 1: if you're separating out code for the sole purpose of readability --- that is, if you could paste the private profile's contents into the main profile for the exact same effect --- you can use a resource-like declaration on the private profile. This lets you consolidate your data lookups and make the private profile's inputs more visible, while keeping the main profile a little cleaner. If you do this, you must make sure to document that the private profile is private.
If there is any chance that this code might be reused by another profile, obey rule 1.

Diff of seventh refactor

@@ -1,8 +1,9 @@
 # /etc/puppetlabs/code/environments/production/site/profile/manifests/
 jenkins/master.pp
 class profile::jenkins::master (  
 - String                      $jenkins_port = '9091',
 - Boolean                     $backups_enabled = false,
 - Boolean                     $manage_plugins = false,
 + Boolean                     $ssl = false,
 + Optional[String[1]]         $site_alias = undef,
   Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
 + Optional[String[1]]         $jenkins_logs_to_syslog = undef,
   Boolean                     $install_jenkins_java = true,
@@ -11,6 +12,9 @@ class profile::jenkins::master (  
 # We rely on virtual resources that are ultimately declared by
- include profile::server.
+ include profile::server
+   # Deploy the SSL certificate/chain/key for sites on this domain.
+   include profile::ssl::delivery_wildcard
+   # Some default values that vary by OS:
+   include profile::jenkins::params
+     $jenkins_owner               = $profile::jenkins::params::jenkins_owner
@@ -22,6 +26,31 @@ class profile::jenkins::master (  
+     # This adds the site_alias to the message of the day for convenience when
+     # logging into a server via FQDN. Because of the way motd::register
+     # works, we need a sort of funny formatting to put it at the end (order => 25) and to
+     # list a class so there isn't a random "--" at the end of the message.
+     if $site_alias {
+       motd::register { 'Jenkins site alias: ${site_alias}'
+         content => @("END"),
+         profile::jenkins::master::proxy
+         Jenkins site alias: ${site_alias}
+         |'-END
+         order => 25,
+       }
+     }
+   + }  
+   + }  
+   + }  
+   + This is a "private" profile that sets up an Nginx proxy -- it's only ever
+   + declared in this class, and it would work identically pasted inline.
+   + But because it's long, this class reads more cleanly with it separated out.
+   class { 'profile::jenkins::master::proxy':
+     site_alias => $site_alias,
+     require_ssl => $ssl,
+     + }
# Sensitive info (like SSH keys) isn't checked into version control like the rest of our modules; instead, it's served from a custom mount point on a designated server.

@@ -56,16 +85,11 @@ class profile::jenkins::master (version            => 'latest',
  service_enable     => true,
  service_ensure     => running,
- configure_firewall => true,
+ configure_firewall => false,
  install_java       => $install_jenkins_java,
  manage_user        => false,
  manage_group       => false,
  manage_datadirs    => false,
- port               => $jenkins_port,
- config_hash        => {
-   'HTTP_PORT'    => { 'value' => $jenkins_port },
-   'JENKINS_PORT' => { 'value' => $jenkins_port },
- },
  }
}

# When not using the jenkins module's java version, install java 8.

The final profile code

After all of this refactoring (and a few more minor adjustments), here's the final code for profile::jenkins::master.

```puppet
# /etc/puppetlabs/code/environments/production/site/profile/manifests/jenkins/master.pp
# Class: profile::jenkins::master
# Install a Jenkins master that meets Puppet's internal needs.

class profile::jenkins::master (
  Boolean $backups_enabled = false,
  Boolean $manage_plugins = false,
  Boolean $ssl = false,
  Optional[String[1]] $site_alias = undef,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
  Optional[String[1]] $jenkins_logs_to_syslog = undef,
  Boolean $install_jenkins_java = true,
) {

  # We rely on virtual resources that are ultimately declared by profile::server.
  include profile::server

  # Deploy the SSL certificate/chain/key for sites on this domain.
  include profile::ssl::delivery_wildcard

  # Some default values that vary by OS:
  include profile::jenkins::params
  $jenkins_owner          = profile::jenkins::params::jenkins_owner
  $jenkins_group          = profile::jenkins::params::jenkins_group
  $master_config_dir      = profile::jenkins::params::master_config_dir

  if $manage_plugins {
    # About 40 jenkins::plugin resources:
    include profile::jenkins::master::plugins
  }
```
motd::register { 'Jenkins CI master (profile::jenkins::master)': }

# This adds the site_alias to the message of the day for convenience when
# logging into a server via FQDN. Because of the way motd::register works,
# we
# need a sort of funny formatting to put it at the end (order => 25) and
to
# list a class so there isn't a random "---" at the end of the message.
if $site_alias {
    motd::register { 'jenkins-site-alias':
        content => @("END",
            profile::jenkins::master::proxy
                Jenkins site alias: ${site_alias}
            |-END
        order => 25,
    }
}

# This is a "private" profile that sets up an Nginx proxy -- it's only
ever
# declared in this class, and it would work identically pasted inline.
# But because it's long, this class reads more cleanly with it separated
out.
class { 'profile::jenkins::master::proxy':
    site_alias => $site_alias,
    require_ssl => $ssl,
}

# Sensitive info (like SSH keys) isn't checked into version control like
the
# rest of our modules; instead, it's served from a custom mount point on a
# designated server.
$secure_server = lookup('puppetlabs::ssl::secure_server')

# Dependencies:
# - Pull in apt if we're on Debian.
# - Pull in the 'git' package, used by Jenkins for Git polling.
# - Manage the 'run' directory (fix for busted Jenkins packaging).
if $::osfamily == 'Debian' { include apt }

package { 'git':
    ensure => present,
}

file { '/var/run/jenkins': ensure => 'directory' }

# Because our account::user class manages the '${master_config_dir}'
directory
# as the 'jenkins' user's homedir (as it should), we need to manage
# '${master_config_dir}/plugins' here to prevent the upstream
# rtyler-jenkins module from trying to manage the homedir as the config
# dir. For more info, see the upstream module's `manifests/plugin.pp`
# manifest.
file { "${master_config_dir}/plugins":
    ensure => directory,
    owner => $jenkins_owner,
    group => $jenkins_group,
    mode => '0755',
    require => [Group[$jenkins_group], User[$jenkins_owner]],
}

Account::User < tag == 'jenkins' >
class { 'jenkins':
  lts        => true,
  repo       => true,
  direct_download => $direct_download,
  version    => 'latest',
  service_enable => true,
  service_ensure => running,
  configure_firewall => false,
  install_java => $install_jenkins_java,
  manage_user => false,
  manage_group => false,
  manage_datadirs => false,
}

# When not using the jenkins module's java version, install java8.
unless $install_jenkins_java { include profile::jenkins::usage::java8 }

# Manage the heap size on the master, in MB.
if($::memorysize_mb =~ Number and $::memorysize_mb > 8192)
{
  # anything over 8GB we should keep max 4GB for OS and others
  $heap = sprintf('%.0f', $::memorysize_mb - 4096)
} else {
  # This is calculated as 50% of the total memory.
  $heap = sprintf('%.0f', $::memorysize_mb * 0.5)
}

# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+j+system+properties
  jenkins::sysconfig { 'JAVA_ARGS':
    value => "-Xms$(heap)m -Xmx$(heap)m -Djava.awt.headless=true
  }

# Forward jenkins master logs to syslog.
# When set to facility.level the jenkins_log uses that value instead of a
# separate log file, for example daemon.info
if $jenkins_logs_to_syslog {
  jenkins::sysconfig { 'JENKINS_LOG':
    value => "$jenkins_logs_to_syslog",
  }
}

# Deploy the SSH keys that Jenkins needs to manage its agent machines and
# access Git repos.
file { "${master_config_dir}/.ssh":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode   => '0700',
}

file { "${master_config_dir}/.ssh/id_rsa":
  ensure => file,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode   => '0600',
  source => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins",
}

file { "${master_config_dir}/.ssh/id_rsa.pub":

© 2022 Puppet, Inc., a Perforce company
Designing convenient roles

There are several approaches to building roles, and you must decide which ones are most convenient for you and your team.

High-quality roles strike a balance between readability and maintainability. For most people, the benefit of seeing the entire role in a single file outweighs the maintenance cost of repetition. Later, if you find the repetition burdensome, you can change your approach to reduce it. This might involve combining several similar roles into a more complex role, creating sub-roles that other roles can include, or pushing more complexity into your profiles.

So, begin with granular roles and deviate from them only in small, carefully considered steps.

Here's the basic Jenkins role we're starting with:

class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}

Related information

Rules for role classes on page 504
There are rules for writing role classes.

**First approach: Granular roles**

The simplest approach is to make one role per type of node, period. For example, the Puppet Release Engineering (RE) team manages some additional resources on their Jenkins masters.

With granular roles, we'd have at least two Jenkins master roles. A basic one:

```erb
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

...and an RE-specific one:

```erb
class role::jenkins::master::release {
  include profile::base
  include profile::server
  include profile::jenkins::master
  include profile::jenkins::master::release
}
```

The benefits of this setup are:

- **Readability** — By looking at a single class, you can immediately see which profiles make up each type of node.
- **Simplicity** — Each role is just a linear list of profiles.

Some drawbacks are:

- **Role bloat** — If you have a lot of only-slightly-different nodes, you quickly have a large number of roles.
- **Repetition** — The two roles above are almost identical, with one difference. If they're two separate roles, it's harder to see how they're related to each other, and updating them can be more annoying.

**Second approach: Conditional logic**

Alternatively, you can use conditional logic to handle differences between closely-related kinds of nodes.

```erb
class role::jenkins::master::release {
  include profile::base
  include profile::server
  include profile::jenkins::master

  if $facts['group'] == 'release' {
    include profile::jenkins::master::release
  }
}
```

The benefits of this approach are:

- You have fewer roles, and they're easy to maintain.

The drawbacks are:

- Reduced readability...maybe. Conditional logic isn't usually hard to read, especially in a simple case like this, but you might feel tempted to add a bunch of new custom facts to accommodate complex roles. This can make roles much harder to read, because a reader must also know what those facts mean.

In short, be careful of turning your node classification system inside-out. You might have a better time if you separate the roles and assign them with your node classifier.
Third approach: Nested roles

Another way of reducing repetition is to let roles include other roles.

```perl
class role::jenkins::master {  
  # Parent role:  
  include role::server  
  # Unique classes:  
  include profile::jenkins::master
}

class role::jenkins::master::release {  
  # Parent role:  
  include role::jenkins::master  
  # Unique classes:  
  include profile::jenkins::master::release
}
```

In this example, we reduce boilerplate by having `role::jenkins::master` include `role::server`. When `role::jenkins::master::release` includes `role::jenkins::master`, it automatically gets `role::server` as well. With this approach, any given role only needs to:

- Include the "parent" role that it most resembles.
- Include the small handful of classes that differentiate it from its parent.

The benefits of this approach are:

- You have fewer roles, and they're easy to maintain.
- Increased visibility in your node classifier.

The drawbacks are:

- Reduced readability: You have to open more files to see the real content of a role. This isn't much of a problem if you go only one level deep, but it can get cumbersome around three or four.

Fourth approach: Multiple roles per node

In general, we recommend that you assign only one role to a node. In an infrastructure where nodes usually provide one primary service, that's the best way to work.

However, if your nodes tend to provide more than one primary service, it can make sense to assign multiple roles.

For example, say you have a large application that is usually composed of an application server, a database server, and a web server. To enable lighter-weight testing during development, you've decided to provide an "all-in-one" node type to your developers. You could do this by creating a new `role::our_application::monolithic` class, which includes all of the profiles that compose the three normal roles, but you might find it simpler to use your node classifier to assign all three roles (`role::our_application::app`, `role::our_application::db`, and `role::our_application::web`) to those all-in-one machines.

The benefit of this approach are:

- You have fewer roles, and they're easy to maintain.

The drawbacks are:

- There's no actual "role" that describes your multi-purpose nodes; instead, the source of truth for what's on them is spread out between your roles and your node classifier, and you must cross-reference to understand their configurations. This reduces readability.
- The normal and all-in-one versions of a complex application are likely to have other subtle differences you need to account for, which might mean making your "normal" roles more complex. It's possible that making a separate role for this kind of node would reduce your overall complexity, even though it increases the number of roles and adds repetition.
Fifth approach: Super profiles
Because profiles can already include other profiles, you can decide to enforce an additional rule at your business: all profiles must include any other profiles needed to manage a complete node that provides that service.

For example, our `profile::jenkins::master` class could include both `profile::server` and `profile::base`, and you could manage a Jenkins master server by directly assigning `profile::jenkins::master` in your node classifier. In other words, a "main" profile would do all the work that a role usually does, and the roles layer would no longer be necessary.

The benefits of this approach are:

- The chain of dependencies for a complex service can be more clear this way.
- Depending on how you conceptualize code, this can be easier in a lot of ways!

The drawbacks are:

- Loss of flexibility. This reduces the number of ways in which your roles can be combined, and reduces your ability to use alternate implementations of dependencies for nodes with different requirements.
- Reduced readability, on a much grander scale. Like with nested roles, you lose the advantage of a clean, straightforward list of what a node consists of. Unlike nested roles, you also lose the clear division between "top-level" complete system configurations (roles) and "mid-level" groupings of technologies (profiles). Not every profile makes sense as an entire system, so you some way to keep track of which profiles are the top-level ones.

Some people really find continuous hierarchies easier to reason about than sharply divided layers. If everyone in your organization is on the same page about this, a "profiles and profiles" approach might make sense. But we strongly caution you against it unless you're very sure; for most people, a true roles and profiles approach works better. Try the well-traveled path first.

Sixth approach: Building roles in the node classifier
Instead of building roles with the Puppet language and then assigning them to nodes with your node classifier, you might find your classifier flexible enough to build roles directly.

For example, you might create a "Jenkins masters" group in the console and assign it the `profile::base`, `profile::server`, and `profile::jenkins::master` classes, doing much the same job as our basic `role::jenkins::master` class.

**Important:**

If you're doing this, make sure you don't set parameters for profiles in the classifier. Continue to use Hiera / Puppet lookup to configure profiles.

This is because profiles are allowed to include other profiles, which interacts badly with the resource-like behavior that node classifiers use to set class parameters.

The benefits of this approach are:

- Your node classifier becomes much more powerful, and can be a central point of collaboration for managing nodes.
- Increased readability: A node's page in the console displays the full content of its role, without having to cross-reference with manifests in your role module.

The drawbacks are:

- Loss of flexibility. The Puppet language's conditional logic is often more flexible and convenient than most node classifiers, including the console.
- Your roles are no longer in the same code repository as your profiles, and it's more difficult to make them follow the same code promotion processes.
Node classifier API v1

These are the endpoints for the node classifier v1 API.

**Tip:** In addition to these endpoints, you can use the status API to check the health of the node classifier service.

- **Forming node classifier requests** on page 529
  Requests to the node classifier API must be well-formed HTTP(S) requests.
- **Groups endpoint** on page 530
  The groups endpoint is used to create, read, update, and delete groups.
- **Groups endpoint examples** on page 548
  Use example requests to better understand how to work with groups in the node classifier API.
- **Classes endpoint** on page 550
  Use the classes endpoints to retrieve lists of classes, including classes with specific environments. The output from this endpoint is especially useful for creating new node groups, which usually contain a reference to one or more classes.
- **Classification endpoint** on page 552
  The classification endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.
- **Commands endpoint** on page 563
  Use the commands endpoint to unpin specified nodes from all groups they’re pinned to.
- **Environments endpoint** on page 564
  Use the environments endpoint to retrieve information about environments in the node classifier. The output tells you either which environments are available or whether a named environment exists. The output can be helpful when creating new node groups, which must be associated with an environment. The node classifier gets its information about environments from Puppet, so do not use this endpoint to create, update, or delete them.
- **Nodes check-in history endpoint** on page 565
  Use the nodes endpoint to retrieve historical information about nodes that have checked into the node classifier.
- **Group children endpoint** on page 567
  Use the group children endpoint to retrieve a specified group and its descendents.
- **Rules endpoint** on page 571
  Use the rules endpoint to translate a group’s rule condition into PuppetDB query syntax.
- **Import hierarchy endpoint** on page 572
  Use the import hierarchy endpoint to delete all existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.
- **Last class update endpoint** on page 573
  Use the last class update endpoint to retrieve the time that classes were last updated from the primary server.
- **Update classes endpoint** on page 573
  Use update classes endpoint to trigger the node classifier to update class and environment definitions from the primary server.
- **Validation endpoints** on page 574
  Use validation endpoints to validate groups in the node classifier.
- **Node classifier errors** on page 577
  Familiarize yourself with error responses to make working the node classifier service API easier.

**Related information**

- **API index** on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Forming node classifier requests**

Requests to the node classifier API must be well-formed HTTP(S) requests.

By default, the node classifier service listens on port 4433 and all endpoints are relative to the `/classifier-api/` path. For example, the full URL for the `/v1/groups` endpoint on localhost would be `https://localhost:4433/classifier-api/v1/groups`.

If needed, you can change the port the classifier API listens on.

**Related information**

[Configuring the console](#) on page 299

After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.

**Authenticating to the node classifier API**

You need to authenticate requests to the node classifier API. You can do this using RBAC authentication tokens or with the list of allowed RBAC certificates.

**Authentication token**

You can make requests to the node classifier API using RBAC authentication tokens.

For detailed information about authentication tokens, see token-based authentication.

In the example, we are using the `/groups` endpoint of the node classifier API to get a list of all groups that exist in the node classifier, along with their associated metadata. The example assumes that you have already generated a token with `puppet-access login`.

```bash
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The example above uses the X-Authentication header to supply the token information. In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter. To supply the token in a token parameter, you would specify the request like this:

```bash
uri="https://$(puppet config print server):4433/classifier-api/v1/groups?
token=$(puppet-access show)"
curl --insecure "$uri"
```

**Note:** Supplying the token as a token parameter is not as secure as using the X-Authentication method.

**Authenticating using an allowed certificate**

You can also authenticate requests using a certificate listed in RBAC's certificate allowlist. The RBAC allowlist is located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. If you edit this file, you must reload the `pe-console-services` service for your changes to take effect (`sudo service pe-console-services reload`). You can attach the certificate using the command line as demonstrated in the example cURL query below. You must have the allowed certificate name and the private key to run the script.

The following query returns a list of all groups that exist in the node classifier, along with their associated metadata. This query shows how to attach the allowlist certificate to authenticate the node classifier API.
In this query, the "allowlisted certname" needs to match a name in the file, /etc/puppetlabs/console-services/rbac-certificate-allowlist.

```bash
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
url="https://$(puppet config print server):4433/classifier-api/v1/groups"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key" "$uri"
```

You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate specifically for use with the API.

**Related information**

- **Token-based authentication** on page 347

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

**Using pagination parameters**

If you have a large number of groups, classes, nodes, node check-ins, or environments, then sending a GET request through the classifier API could return an excessively large number of items.

To limit the number of items returned, you can use the `limit` and `offset` parameters.

- **limit**: The value of the `limit` parameter limits how many items are returned in a response. The value must be a non-negative integer. If you specify a value other than a non-negative integer, you get a 400 Bad Request error.
- **offset**: The value of the `offset` parameter specifies the number of item that are skipped. For example, if you specify an offset of 20 with a limit of 10, as shown in the example below, the first 20 items are skipped and you get back item 21 through to item 30. The value must be a non-negative integer. If you specify a value other than a non-negative integer, you get a 400 Bad Request error.

The following example shows a request using the `limit` and `offset` parameters.

```bash
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
url="https://$(puppet config print server):4433/classifier-api/v1/groups?limit=10&offset=20"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

**Groups endpoint**

The **groups** endpoint is used to create, read, update, and delete groups.

A group belongs to an environment, applies classes (possibly with parameters), and matches nodes based on rules. Because groups are so central to the classification process, this endpoint is where most of the action is.

**GET /v1/groups**

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.
Query parameters

The request accepts these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inherited</td>
<td>If set to any value besides 0 or false, the node group includes the classes, class parameters, configuration data, and variables that it inherits from its ancestors.</td>
</tr>
</tbody>
</table>

Response format

The response is a JSON array of node group objects, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (a string).</td>
</tr>
<tr>
<td>id</td>
<td>The node group's ID, which is a string containing a type-4 (random) UUID. The regular expression used to validate node group UUIDs is <code>[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}</code>.</td>
</tr>
<tr>
<td>description</td>
<td>An optional key containing an arbitrary string describing the node group.</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group, which is the root of the hierarchy. Note that the root group always has the lowest-possible random UUID, 00000000-0000-4000-8000-000000000000.</td>
</tr>
<tr>
<td>rule</td>
<td>A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>classes</td>
<td>An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects’ keys are parameter names, and the values are what the node group sets for that parameter (always a string).</td>
</tr>
<tr>
<td>config_data</td>
<td>An object similar to the classes object that specifies parameters that are applied to classes if the class is assigned in the classifier or in puppet code. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects’ keys are parameter names, and the values are what the group sets for that parameter (always a string). This feature is enabled/disabled via the classifier::allow-config-data setting. When set to false, this key is stripped from the payload.</td>
</tr>
<tr>
<td>deleted</td>
<td>An object similar to the classes object that shows which classes and class parameters set by the node group have since been deleted. If none of the node group’s classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the puppetlabs.classifier/deleted key, whose value is a Boolean indicating whether the entire class has been deleted. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: puppetlabs.classifier/deleted, mapping to a boolean indicating whether the specific class parameter has been deleted; and value, mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the classes object).</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).</td>
</tr>
<tr>
<td>last_edited</td>
<td>The most recent time at which a user committed changes to a node group. This is a time stamp in ISO 8601 format, YYYY-MM-DDTHH:MM:SSZ.</td>
</tr>
<tr>
<td>serial_number</td>
<td>A number assigned to a node group. This number is incremented each time changes to a group are committed. serial_number is used to prevent conflicts when multiple users make changes to the same node group at the same time.</td>
</tr>
</tbody>
</table>
This example shows a node group object:

```json
{
  "name": "Webservers",
  "id": "fc500c43-5065-469b-91fc-37ed0e500e81",
  "last_edited": "2018-02-20T02:36:17.776Z",
  "serial_number": 16,
  "environment": "production",
  "description": "This group captures configuration relevant to all web-facing production webservers, regardless of location.",
  "parent": "00000000-0000-4000-8000-000000000000",
  "rule": ["and", ["~", ["trusted", "certname"], "www"], [">=", ["fact", "total_ram"], "512"]],
  "classes": {
    "apache": {
      "serveradmin": "bofh@travaglia.net",
      "keepalive_timeout": "5"
    },
  },
  "config_data": {
    "puppet_enterprise::profile::console": {"certname": "console.example.com"},
    "puppet_enterprise::profile::puppetdb": {"listen_address": "0.0.0.0"}
  },
  "variables": {
    "ntp_servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org", "2.us.pool.ntp.org"]
  }
}
```

This example shows a node group object that refers to some classes and parameters that have been deleted:

```json
{
  "name": "Spaceship",
  "id": "fc500c43-5065-469b-91fc-37ed0e500e81",
  "last_edited": "2018-03-13T21:37:03.608Z",
  "serial_number": 42,
  "environment": "space",
  "parent": "00000000-0000-4000-8000-000000000000",
  "rule": ["=", ["fact", "is_spaceship"], "true"],
  "classes": {
    "payload": {
      "type": "cubesat",
      "count": "8",
      "mass": "10.64"
    },
    "rocket": {
      "stages": "3"
    }
  },
  "deleted": {
    "payload": {"puppetlabs.classifier/deleted": true},
    "rocket": {
      "puppetlabs.classifier/deleted": false,
      "stages": {
        "puppetlabs.classifier/deleted": true,
        "value": "3"
      }
    }
  },
  "variables": {}
The entire payload class has been deleted, because its deleted parameters object’s puppetlabs.classifier/deleted key maps to true, in contrast to the rocket class, which has had only its stages parameter deleted.

Rule condition grammar

Nodes can be classified into groups using rules. This example shows how rule conditions must be structured:

```plaintext
condition : [ {bool} {condition}+ ] | [ "not" {condition} ] | {operation}

  bool : "and" | "or"

  operation : [= | "~" | > | >= | < | <= ]

  fact-path : {field-name} | [ {path-type} {field-name} {path-component}+ ]

  path-type : "trusted" | "fact"

  path-component : field-name | number

  field-name : string
```

For the regex operator "~", the value is interpreted as a Java regular expression. Literal backslashes must be used to escape regex characters in order to match those characters in the fact value.

For the numeric comparison operators (">", ">=", "<", and "<="), the fact value (which is always a string) is coerced to a number (either integral or floating-point). If the value can’t be coerced to a number, the numeric operation evaluates to false.

For the fact path, the rule can be either a string representing a top level field (the only current meaningful value here would be "name" representing the node name) or a list of strings and indices that represent looking up a field in a nested data structure. When passing a list of strings or indices, the first and second entries in the list must be strings. Subsequent entries can be indices.

Regular facts start with "fact" (for example, ["fact", "architecture"])) and trusted facts start with "trusted" (for example, ["trusted", "certname"]).

Error responses

serial_number

If you commit a node group with a serial_number that an API call has previously assigned, the service returns a 409 Conflict response.

POST /v1/groups

Use the /v1/groups endpoint to create a new node group without specifying its ID. When you use this endpoint, the node classifier service randomly generates an ID.

Request format

The request body must be a JSON object describing the node group to be created. The request uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (a string).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group’s environment. This key is optional; if it’s not provided, the default environment (production) is used.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>Whether this node group’s environment should override those of other node groups at classification-time. This key is optional; if it’s not provided, the default value of false is used.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (required).</td>
</tr>
<tr>
<td>rule</td>
<td>The condition that must be satisfied for a node to be classified into this node group</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The variables key is optional, and if a node group does not define any top-level variables then it may be omitted.</td>
</tr>
<tr>
<td>classes</td>
<td>An object that defines the classes to be used by nodes in the node group. The classes key is required, and at minimum should be an empty object ({}). The classes key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The classes key is not optional; if it is missing, the service returns a 400 Bad Request response.</td>
</tr>
<tr>
<td>config_data</td>
<td>An optional object that defines the class parameters to be used by nodes in the group. Its structure is the same as the classes object. If you use a config_data key but provide only the class, like &quot;config_data&quot;: {&quot;qux&quot;:{}}, no configuration data is stored.</td>
</tr>
</tbody>
</table>

**Note:** This feature is enabled with the classifier::allow-config-data setting. When set to false, the presence of this key in the payload results in a 400 response.

**Response format**

If the node group was successfully created, the service returns a 303 See Other response, with the path to retrieve the created node group in the "location" header of the response.

**Error responses**

Responses and keys returned for create group requests depend on the type of error.

schema-violation

© 2022 Puppet, Inc., a Perforce company
If any of the required keys are missing or the values of any of the defined keys do not match the required type, the service returns a **400 Bad Request** response using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;schema-violation&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An object that contains three keys:</td>
</tr>
<tr>
<td></td>
<td>• submitted — Describes the submitted object.</td>
</tr>
<tr>
<td></td>
<td>• schema — Describes the schema that object is</td>
</tr>
<tr>
<td></td>
<td>expected to conform to.</td>
</tr>
<tr>
<td></td>
<td>• error — Describes how the submitted object</td>
</tr>
<tr>
<td></td>
<td>failed to conform to the schema.</td>
</tr>
</tbody>
</table>

### malformed-request

If the request's body could not be parsed as JSON, the service returns a **400 Bad Request** response using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-request&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An object that contains two keys:</td>
</tr>
<tr>
<td></td>
<td>• body — Holds the request body that was received.</td>
</tr>
<tr>
<td></td>
<td>• error — Describes how the submitted object</td>
</tr>
<tr>
<td></td>
<td>failed to conform to the schema.</td>
</tr>
</tbody>
</table>

### uniqueness-violation

If your attempt to create the node group violates uniqueness constraints (such as the constraint that each node group name must be unique within its environment), the service returns a **422 Unprocessable Entity** response using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;uniqueness-violation&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Describes which fields of the node group causd</td>
</tr>
<tr>
<td></td>
<td>the constraint to be violated, along with their</td>
</tr>
<tr>
<td></td>
<td>values.</td>
</tr>
<tr>
<td>details</td>
<td>An object that contains two keys:</td>
</tr>
<tr>
<td></td>
<td>• conflict — An object whose keys are the fields</td>
</tr>
<tr>
<td></td>
<td>of the node group that violated the constraint</td>
</tr>
<tr>
<td></td>
<td>and whose values are the corresponding field</td>
</tr>
<tr>
<td></td>
<td>values.</td>
</tr>
<tr>
<td></td>
<td>• constraintName — The name of the database</td>
</tr>
<tr>
<td></td>
<td>constraint that was violated.</td>
</tr>
</tbody>
</table>

### missing-referents
If classes or class parameters defined by the node group, or inherited by the node group from its parent, do not exist in the submitted node group's environment, the service returns a 422 Unprocessable Entity response. In both cases the response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;missing-referents&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Describes the error and lists the missing classes or parameters.</td>
</tr>
<tr>
<td>details</td>
<td>An array of objects, where each object describes a single missing referent, and has the following keys:</td>
</tr>
<tr>
<td></td>
<td>• kind — &quot;missing-class&quot; or &quot;missing-parameter&quot;, depending on whether the entire class doesn't exist, or the class just doesn't have the parameter.</td>
</tr>
<tr>
<td></td>
<td>• missing — The name of the missing class or class parameter.</td>
</tr>
<tr>
<td></td>
<td>• environment — The environment that the class or parameter is missing from; that is, the environment of the node group where the error was encountered.</td>
</tr>
<tr>
<td></td>
<td>• group — The name of the node group where the error was encountered. Due to inheritance, this might not be the group where the parameter was defined.</td>
</tr>
<tr>
<td></td>
<td>• defined_by — The name of the node group that defines the class or parameter.</td>
</tr>
</tbody>
</table>

missing-parent

If the parent of the node group does not exist, the service returns a 422 Unprocessable Entity response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;missing-parent&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Shows the parent UUID that did not exist.</td>
</tr>
<tr>
<td>details</td>
<td>The full submitted node group.</td>
</tr>
</tbody>
</table>

inheritance-cycle

If the request causes an inheritance cycle, the service returns a 422 Unprocessable Entity response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;inheritance-cycle&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An array of node group objects that includes each node group involved in the cycle</td>
</tr>
</tbody>
</table>
**Key** | **Definition**
--- | ---
msg | A shortened description of the cycle, including a list of the node group names with each followed by its parent until the first node group is repeated.

**GET /v1/groups/<id>**

Use the `/v1/groups/<id>` endpoint to retrieve a node group with the given ID.

**Response format**

The response is a JSON array of node group objects, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (a string).</td>
</tr>
<tr>
<td>id</td>
<td>The node group's ID, which is a string containing a type-4 (random) UUID. The regular expression used to validate node group UUIDs is `^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$.</td>
</tr>
<tr>
<td>description</td>
<td>An optional key containing an arbitrary string describing the node group.</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group, which is the root of the hierarchy. Note that the root group always has the lowest-possible random UUID, <code>00000000-0000-4000-8000-000000000000</code>.</td>
</tr>
<tr>
<td>rule</td>
<td>A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>classes</td>
<td>An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the node group sets for that parameter (always a string).</td>
</tr>
<tr>
<td>deleted</td>
<td>An object similar to the classes object that shows which classes and class parameters set by the node group have since been deleted. If none of the node group's classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the puppetlabs.classifier/deleted key, whose value is a Boolean indicating whether the entire class has been deleted. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: puppetlabs.classifier/deleted, mapping to a boolean indicating whether the specific class parameter has been deleted; and value, mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the classes object).</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).</td>
</tr>
<tr>
<td>last_edited</td>
<td>The most recent time at which a user committed changes to a node group. This is a time stamp in ISO 8601 format, YYYY-MM-DDTHH:MM:SSZ.</td>
</tr>
<tr>
<td>serial_number</td>
<td>A number assigned to a node group. This number is incremented each time changes to a group are committed. serial_number is used to prevent conflicts when multiple users make changes to the same node group at the same time.</td>
</tr>
</tbody>
</table>

**Error responses**

If the node group with the given ID cannot be found, the service returns 404 Not Found and malformed-uuid responses. The body includes a generic 404 error response as described in the errors documentation.

**serial_number**

If you commit a node group with a serial_number that an API call has previously assigned, the service returns a 409 Conflict response.
Related information

Node classifier errors on page 577
Familiarize yourself with error responses to make working the node classifier service API easier.

**PUT /v1/groups/<id>**

Use the `/v1/groups/<id>` to create a node group with the given ID.

⚠️ **CAUTION:** Any existing node group with the given ID is overwritten.
It is possible to overwrite an existing node group with a new node group definition that contains deleted classes or parameters.

**Important:** You must supply a valid type-4 (random) UUID for the ID.

### Request format

The request body must be a JSON object describing the node group to be created. The request uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (a string).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group's environment. This key is optional; if it's not provided, the default environment (production) is used.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>Whether this node group's environment should override those of other node groups at classification-time. This key is optional; if it's not provided, the default value of false is used.</td>
</tr>
<tr>
<td>description</td>
<td>A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (required).</td>
</tr>
<tr>
<td>rule</td>
<td>The condition that must be satisfied for a node to be classified into this node group</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The variables key is optional, and if a node group does not define any top-level variables then it may be omitted.</td>
</tr>
</tbody>
</table>
### Key Definition

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>classes</td>
<td>An object that defines the classes to be used by nodes in the node group. The <code>classes</code> key is required, and at minimum should be an empty object ({}). The <code>classes</code> key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The <code>classes</code> key is <em>not</em> optional; if it is missing, the service returns a 400 <code>Bad Request</code> response.</td>
</tr>
</tbody>
</table>

### Response format

If the node group is successfully created, the service returns a 201 `Created` response, with the node group object (in JSON) as the body. If the node group already exists and is identical to the submitted node group, then the service takes no action and returns a 200 `OK` response, again with the node group object as the body.

### Error responses

If the requested node group object contains the `id` key, and its value differs from the UUID specified in the request's path, the service returns a 400 `Bad Request` response.

The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;conflicting-ids&quot;</td>
</tr>
</tbody>
</table>
| details | An object that contains two keys:  
  • `submitted`— Contains the ID submitted in the request body.  
  • `fromUrl`— Contains the ID taken from the request URL. |

In addition, this operation can produce the general `malformed-error` response and any response that could also be generated by the POST group creation endpoint.

**POST /v1/groups/<id>**

Use the `/v1/groups/<id>` endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

### Request format

The request body must be JSON object describing the delta to be applied to the node group.

The `classes`, `config_data`, `variables`, and `rule` keys of the delta are merged with the node group, and then any keys of the resulting object that have a null value are deleted. This allows you to remove classes, class parameters, configuration data, variables, or the rule from the node group by setting them to null in the delta.
If the delta has a rule key that’s set to a new value or nil, it’s updated wholesale or removed from the group accordingly.

The name, environment, description, and parent keys, if present in the delta, replace the old values wholesale with their values.

The serial_number key is optional. If you update a node group and provide the serial_number in the payload, and the serial_number is not the current one for that group, the service returns a 409 Conflict response. To bypass this check, omit the serial_number.

Note that the root group’s rule cannot be edited; any attempts to do so raise a 422 Unprocessable Entity response.

In the following examples, a delta is merged with a node group to update the group.

Node group:

```json
{
   "name": "Webservers",
   "id": "58463036-0efa-4365-b367-b5401c0711d3",
   "environment": "staging",
   "parent": "00000000-0000-4000-8000-000000000000",
   "rule": ["~", ["trusted", "certname"], "www"],
   "classes": {
      "apache": {
         "serveradmin": "bofh@travaglia.net",
         "keepalive_timeout": 5
      },
      "ssl": {
         "keystore": "/etc/ssl/keystore"
      }
   },
   "variables": {
      "ntp_servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org", "2.us.pool.ntp.org"]
   }
}
```

Delta:

```json
{
   "name": "Production Webservers",
   "id": "58463036-0efa-4365-b367-b5401c0711d3",
   "environment": "production",
   "parent": "01522c99-627c-4a07-b28e-a25dd563d756",
   "classes": {
      "apache": {
         "serveradmin": "roy@reynholm.co.uk",
         "keepalive_timeout": null
      },
      "ssl": null
   },
   "variables": {
      "dns_servers": ["dns.reynholm.co.uk"]
   }
}
```

Updated group:

```json
{
   "name": "Production Webservers",
   "id": "58463036-0efa-4365-b367-b5401c0711d3",
   "environment": "production",
   "rule": ["~", ["trusted", "certname"], "www"],
   "classes": {
      "apache": {
         "serveradmin": "bofh@travaglia.net",
         "keepalive_timeout": 5
      },
      "ssl": {
         "keystore": "/etc/ssl/keystore"
      }
   },
   "variables": {
      "ntp_servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org", "2.us.pool.ntp.org"]
   }
}
```
"parent": "01522c99-627c-4a07-b28e-a25dd563d756",
"rule": ["~", ["trusted", "certname"], "www"],
"classes": {
  "apache": {
    "serveradmin": "roy@reynholm.co.uk"
  }
},
"variables": {
  "ntp_servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org",
  "2.us.pool.ntp.org"],
  "dns_servers": ["dns.reynholm.co.uk"]
}
}

Note that the ssl class was deleted because its entire object was mapped to null, whereas for the apache class only the keepalive_timeout parameter was deleted.

### Deleted classes and class parameters

If the node group definition contains classes and parameters that have been deleted it is still possible to update the node group with those parameters and classes. Updates that don’t increase the number of errors associated with a node group are allowed.

### Error responses

The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;conflicting-ids&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An object that contains two keys:</td>
</tr>
<tr>
<td></td>
<td>• submitted — Contains the ID submitted in the request body.</td>
</tr>
<tr>
<td></td>
<td>• fromUrl — Contains the ID taken from the request URL.</td>
</tr>
</tbody>
</table>

If the requested node group object contains the id key, and its value differs from the UUID specified in the request’s path, the service returns a 400 Bad Request response.

In addition, this operation can produce the general malformed-error response and any response that could also be generated by the POST group creation endpoint.

422 responses to POST requests can include errors that were caused by the node group’s children, but a node group being created with a PUT request cannot have any children.

**DELETE /v1/groups/<id>**

Use the /v1/groups/\<id\> endpoint to delete the node group with the given ID.

### Response format

If the delete operation is successful, the sever returns a 204 No Content response.

### Error responses

In addition to the general malformed-uuid response, if the node group with the given ID does not exist, the service returns a 404 Not Found response, as described in the errors documentation.
children-present

The service returns a 422 Unprocessable Entity and rejects the delete request if the node group that is being deleted has children.

The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;children-present&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Explains why the delete was rejected and names the children.</td>
</tr>
<tr>
<td>details</td>
<td>Contains the node group in question along with all of its children.</td>
</tr>
</tbody>
</table>

Related information

Node classifier errors on page 577

Familiarize yourself with error responses to make working the node classifier service API easier.

POST /v1/groups/<id>/pin

Use the /v1/groups/<id>/pin endpoint to pin nodes to the group with the given ID.

Request format

You can provide the names of the nodes to pin in two ways.

- As the value of the nodes query parameter. For multiple nodes, use a comma-separated list format.
  
  For example:
  
  POST /v1/groups/58463036-0efa-4365-b367-b5401c0711d3/pin?nodes=foo%2Cbar%2Cbaz
  
  This request pins the nodes foo, bar, and baz to the group.

- In the body of a request. In the nodes field of a JSON object, specify the node name as the value. For multiple nodes, use a JSON array.

  For example:

  {
  "nodes": ["foo", "bar", "baz"]
  }

  This request pins the nodes foo, bar, and baz to the group.

It's easier to use the query parameter method. However, if you want to affect a large number of nodes at once, the query string might get truncated. Strings are truncated if they exceed 8,000 characters. In such cases, use the second method. The request body in the second method is allowed to be many megabytes in size.

Response format

If the pin is successful, the service returns a 204 No Content response with an empty body.

Error responses

This endpoint uses the following error responses.

If you don't supply the nodes query parameter or a request body, the service returns a 400 Malformed Request response, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;missing-parameters&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Explains the missing nodes query parameter.</td>
</tr>
</tbody>
</table>
If you supply a request body that is not valid JSON, the service returns a 400 Malformed Request response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-request&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An object with a body key containing the body as</td>
</tr>
<tr>
<td></td>
<td>received by the service, and an error field containing</td>
</tr>
<tr>
<td></td>
<td>a string describing the error encountered when trying to</td>
</tr>
<tr>
<td></td>
<td>parse the request's body.</td>
</tr>
</tbody>
</table>

If the request's body is valid JSON, but the payload is not an object with just the nodes field and no other fields, the service returns a 400 Malformed Request response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;schema-violation&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Describes the difference between what was submitted</td>
</tr>
<tr>
<td></td>
<td>and the required format.</td>
</tr>
</tbody>
</table>

POST /v1/groups/<id>/unpin

Use the /v1/groups/<id>/unpin endpoint to unpin nodes from the group with the given ID.

**Note:** Nodes not actually pinned to the group can be specified without resulting in an error.

**Request format**

You can provide the names of the nodes to pin in two ways.

- As the value of the nodes query parameter. For multiple nodes, use a comma-separated list format.

  For example:

  POST /v1/groups/58463036-0efa-4365-b367-b5401c0711d3/unpin?nodes=foo%2Cbar%2Cbaz

  This request unpins the nodes foo, bar, and baz from the group. If any of the specified nodes were not pinned to the group, they are ignored.

- In the body of a request. In the nodes field of a JSON object, specify the node name as the value. For multiple nodes, use a JSON array.

  For example:

  ```json
  {"nodes": ["foo", "bar", "baz"]}
  ```

  This request unpins the nodes foo, bar, and baz from the group. If any of the specified nodes were not pinned to the group, they are ignored.

It's easier to use the query parameter method. However, if you want to affect a large number of nodes at once, the query string might get truncated. Strings are truncated if they exceed 8,000 characters. In such cases, use the second method. The request body in the second method is allowed to be many megabytes in size.

**Response format**

If the unpin is successful, the service returns a 204 No Content response with an empty body.

**Error responses**

If you don't supply the nodes query parameter or a request body, the service returns a 400 Malformed Request response. The response object uses these keys:
### Key Definition

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;missing-parameters&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Explains the missing nodes query parameter.</td>
</tr>
</tbody>
</table>

If a request body is supplied but it is not valid JSON, the service returns a 400 Malformed Request response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-request&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An object with a body key containing the body as received by the service, and an error field containing a string describing the error encountered when trying to parse the request’s body.</td>
</tr>
</tbody>
</table>

If the request’s body is valid JSON, but the payload is not an object with just the nodes field and no other fields, the service returns a 400 Malformed Request response. The response object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;schema-violation&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Describes the difference between what was submitted and the required format.</td>
</tr>
</tbody>
</table>

### GET /v1/groups/:id/rules

Use /v1/groups/:id/rules to resolve the rules for the requested group and then translate those rules to work with the nodes and inventory endpoints in PuppetDB.

**Response format**

A successful response includes these keys:

- **rule**
  The rules for the group in classifier format.

- **rule_with_inherited**
  The inherited rules (including the rules for this group) in classifier format

- **translated**
  An object containing two children, each of the inherited rules translated into a different format.

  - **nodes_query_format**
    The optimized translated inherited group in the format that works with the nodes endpoint in PuppetDB.

  - **inventory_query_format**
    The optimized translated inherited group in the format that works with the inventory endpoint in PuppetDB.

```json
{
  "rule": [
    ["=",
    ["fact",
     "is_spaceship"],
    "true"],
  "rule_with_inherited": [
    "and",
```
Error responses

In addition to the general malformed-uuid error response, if the group with the given ID cannot be found, a 404 Not Found response is returned.

Related information
Error response description on page 578
Errors from the node classifier service are JSON responses.

**Groups endpoint examples**

Use example requests to better understand how to work with groups in the node classifier API.

These requests assume the following configuration:

- The Puppet primary server is running on puppetlabs-nc.example.vm and can allow certificates to enable URL requests.
- Port 4433 is open.

**Create a group called My Nodes**

This request uses POST /v1/groups to create a group called *My Nodes*.

```bash
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
data='{
  "name": "My Nodes",
  "parent": "00000000-0000-4000-8000-000000000000",
  "environment": "production",
  "classes": {}}
'
curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
--request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

**Related information**

**POST /v1/groups** on page 534

Use the /v1/groups endpoint to create a new node group without specifying its ID. When you use this endpoint, the node classifier service randomly generates an ID.

**Get the group ID of My Nodes**

This request uses the groups endpoint to get details about groups.

```bash
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
"$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The response is a JSON file containing details about all groups, including the *My Nodes* group ID.

```json
{
  "environment_trumps": false,
  "parent": "00000000-0000-4000-8000-000000000000",
  "name": "My Nodes",
  "variables": {},
  "id": "085e2797-32f3-4920-9412-8e9decf4ef65",
  "environment": "production",
  "classes": {}}
```
Related information

Groups endpoint on page 530
The groups endpoint is used to create, read, update, and delete groups.

Get a list of all node groups and their IDs
Use this curl command to get a list of all node groups and their corresponding IDs. Node group IDs are useful when running tasks or plans.

```bash
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
curl --silent --header "$auth_header" "$uri" | jq -M -r '.[].name
(.id)

See Usage notes for curl examples for information about forming curl commands.

The response is a list of all node groups and their IDs, for example:

All Nodes 0000000-0000-4000-8000-000000000000
PE Master 07002034-c20f-44de-97d2-f72d834e81fb
Development one-time run exception 124a11d8-b912-45f0-9a6d-5ddd81aa0ed
PE PuppetDB 289f176b-1c30-4e85-ad07-131e55f29354
PE Database 28b78e75-3b7e-464e-8f02-29a80b88fe02
Production environment 3882e9a-2f91-4ed7-8f84-93d8bf115ec5
PE Orchestrator 3f490039-395f-4c87-8dfb-f72d034e81fb
All Environments 6a10e0eb-ab6b-4ba7-b637-28df91ed659
PE Compiler 9cab6f77-40cf-40e-b2ce-6aa6a2489c71
PE Infrastructure 9cd74d7b-6fb7-4d17-9cd8-3ef743989a5c
PE Patch Management aa9e4cd-fed5-4f07-8149-98a699a3b92
PE Certificate Authority d406a750-0cab-43cf-a4fa-93d8bf115ec5
PE Agent d4cf6659-6fc8-4419-b2a2-28df91ed659
PE Console de97e269-4a9e-4f3d-a931-39993b0ef3f6
PE Infrastructure Agent e46543a6-61c6-49f2-865f-28df91ed659

Pin a node to the My Nodes group
This request uses POST /v1/groups/<id> to pin the node example-to-pin.example.vm to My Groups using the group ID retrieved in the previous step.

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin"
data='{"nodes": ["example-to-pin.example.vm"] }'
curl --header "$type_header" --header "$auth_header" --request POST "$uri"
--data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

Related information

POST /v1/groups/<id> on page 541
Use the /v1/groups/<id> endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Pin a second node to the My Nodes group
This request uses POST /v1/groups/<id> to pin a second node, example-to-pin-2.example.vm, to My Groups.

You only need to include new nodes, although it does not hurt to include previously-pinned nodes.

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
```
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin"
data='{"nodes": ["example-to-pin-2.example.vm"] }'

curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"

See Usage notes for curl examples for information about forming curl commands.

Related information

POST /v1/groups/<id> on page 541

Use the /v1/groups/<id> endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Unpin a node from the My Nodes group

This request uses POST /v1/groups/<id> to unpin the node example-to-unpin.example.vm from My Groups.

type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/a02ee4a7-8ec1-44ec-99a6-e362596fd6e/unpin"
data='{"nodes": ["example-to-unpin"] }'

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key" --request POST "$uri" --data "$data"

See Usage notes for curl examples for information about forming curl commands.

Add a class and parameters to the My Nodes group

This request uses POST /v1/groups/<id> to specify the apache class and parameters for My Groups.

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65"
data='{"classes":
   {"apache": {
      "serveradmin": "bob@example.com",
      "keepalive_timeout": null
   }
   }
}"
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"

See Usage notes for curl examples for information about forming curl commands.

Related information

POST /v1/groups/<id> on page 541

Use the /v1/groups/<id> endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Classes endpoint

Use the classes endpoints to retrieve lists of classes, including classes with specific environments. The output from this endpoint is especially useful for creating new node groups, which usually contain a reference to one or more classes.

The node classifier gets its information about classes from Puppet, so don't use this endpoint to create, update, or delete classes.
**GET /v1/classes**

Use the `/v1/classes` endpoint to retrieve a list of all classes known to the node classifier.

**Note:** All other operations on classes require using the environment-specific endpoints.

All `/classes` endpoints return the classes *currently known* to the node classifier, which retrieves them periodically from the primary server. To force an update, use the `update_classes` endpoint. To determine when classes were last retrieved from the primary server, use the `last_class_update` endpoint.

**Response format**

The response is a JSON array of class objects. Each class object contains these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the class (a string).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the environment that this class exists in.</td>
</tr>
<tr>
<td>parameters</td>
<td>An object describing the parameters and default parameter values for the class. Each key is the parameter name (string). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or <code>null</code>. If the value is <code>null</code>, the parameter is required.</td>
</tr>
</tbody>
</table>

This is an example of a class object:

```
{
    "name": "apache",
    "environment": "production",
    "parameters": {
        "default_mods": true,
        "default_vhost": true,
        ...
    }
}
```

**GET /v1/environments/<environment>/classes**

Use the `/v1/environments/<environment>/classes` to retrieve a list of all classes known to the node classifier within the given environment.

**Response format**

The response is a JSON array of class objects. Each class object contains these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the class (a string).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the environment that this class exists in.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>An object describing the parameters and default parameter values for the class. The keys of this object are the parameter names (strings). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or null. If the value is null, the parameter is required.</td>
</tr>
</tbody>
</table>

**Error responses**

This request does not produce error responses.

**GET /v1/environments/<environment>/classes/<name>**

Use the `/v1/environments/<environment>/classes/<name>` endpoint to retrieve the class with the given name in the given environment.

**Response format**

If the class exists, the response is a JSON array of class objects. Each class object contains these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the class (a string).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the environment that this class exists in. Note that the same class can exist in different environments and can have different parameters in each environment.</td>
</tr>
<tr>
<td>parameters</td>
<td>An object describing the parameters and default parameter values for the class. The keys of this object are the parameter names (strings). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or null. If the value is null, the parameter is required.</td>
</tr>
</tbody>
</table>

**Error responses**

If the class with the given name cannot be found, the service returns a 404 *Not Found* response with an empty body.

**Classification endpoint**

The classification endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

**POST /v1/classified/nodes/<name>**

Use the `/v1/classified/nodes/<name>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

**Request format**

The request body can contain a JSON object describing the facts and trusted facts of the node to be classified. The object can have these keys:
### Key Definition

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact</td>
<td>The regular, non-trusted facts of the node. The value of this key is a further object, whose keys are fact names, and whose values are the fact values. Fact values can be a string, number, boolean, array, or object.</td>
</tr>
<tr>
<td>trusted</td>
<td>The trusted facts of the node. The values of this key are subject to the same restrictions as those on the value of the fact key.</td>
</tr>
</tbody>
</table>

### Response format

The response is a JSON object describing the node post-classification, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node (a string).</td>
</tr>
<tr>
<td>groups</td>
<td>An array of the group-ids (strings) that this node was classified into.</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the environment that this node uses, which is taken from the node groups the node was classified into.</td>
</tr>
<tr>
<td>classes</td>
<td>An object where the keys are class names and the values are objects that map parameter names to values.</td>
</tr>
<tr>
<td>parameters</td>
<td>An object where the keys are top-level variable names and the values are the values assigned to those variables.</td>
</tr>
</tbody>
</table>

This is an example of a response from this endpoint:

```json
{
    "name": "foo.example.com",
    "groups": ["9c07d07-a199-48b7-9999-3c07d07-a0bf", "96d1a058-225d-48e2-a1a8-80819d31751d"],
    "environment": "staging",
    "parameters": {},
    "classes": {
        "apache": {
            "keepalive_timeout": 30,
            "log_level": "notice"
        }
    }
}
```

### Error responses

If the node is classified into multiple node groups that define conflicting classifications for the node, the service returns a 500 Server Error response.

The body of this response contains the usual JSON error object described in the errors documentation.

The kind key of the error is "classification-conflict", the msg describes generally why this happens, and the details key contains an object that describes the specific conflicts encountered.

The details object can have these keys:
### Key | Definition
--- | ---
environment | Maps directly to an array of value detail objects (described below).
variables | Contains an object with a key for each conflicting variable, whose values are an array of value detail objects.
classes | Contains an object with a key for each class that had conflicting parameter definitions, whose values are further objects that describe the conflicts for that class's parameters.

A value details object describes one of the conflicting values defined for the environment, a variable, or a class parameter. Each object contains these keys:

### Key | Definition
--- | ---
**value** | The defined value, which is a string for environment and class parameters, but for a variable can be any JSON value.
**from** | The node group that the node was classified into that caused this value to be added to the node's classification. This group cannot define the value, because it can be inherited from an ancestor of this group.
**defined_by** | The node group that actually defined this value. This is often the `from` group, but could instead be an ancestor of that group.

This example shows a classification conflict error object with node groups truncated for clarity:

```json
{
  "kind": "classification-conflict",
  "msg": "The node was classified into multiple unrelated node groups that defined conflicting class parameters or top-level variables. See `details` for a list of the specific conflicts.",
  "details": {
    "classes": {
      "songColors": {
        "blue": [
          {
            "value": "Blue Suede Shoes",
            "from": {
              "name": "Elvis Presley",
              "classes": {},
              "rule": ["==", "nodename", "the-node"],
              ...
            },
            "defined_by": {
              "name": "Carl Perkins",
              "classes": {
                "songColors": {
                  "blue": "Blue Suede Shoes"},
                "rule": ["not", ["==", "nodename", "the-node"],
                ...
              }},
            },
          {
            "value": "Since You've Been Gone",
            "from": {
              "name": "Aretha Franklin",
              ...
            },
          }
        ]
      }
    }
  }
}
```

© 2022 Puppet, Inc., a Perforce company
In this example, the conflicting "Blue Suede Shoes" value was included in the classification because the node matched the "Elvis Presley" node group (because that is the value of the "from" key), but that node group doesn't define the "Blue Suede Shoes" value. That value is defined by the "Carl Perkins" node group, which is an ancestor of the "Elvis Presley" node group, causing the latter to inherit the value from the former. The other conflicting value, "Since You've Been Gone", is defined by the same node group that the node matched.

Related information
Node classifier errors on page 577
Familiarize yourself with error responses to make working the node classifier service API easier.

POST /v1/classified/nodes/<name>/explanation
Use the /v1/classified/nodes/<name>/explanation endpoint to retrieve an explanation of how a node is classified by submitting its facts.

Request format
The body of the request must be a JSON object describing the node's facts. This object uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact</td>
<td>Describes the regular (non-trusted) facts of the node. Its value must be a further object whose keys are fact names, and whose values are the corresponding fact values. Structured facts can be included here; structured fact values are further objects or arrays.</td>
</tr>
<tr>
<td>trusted</td>
<td>Optional key that describes the trusted facts of the node. Its value has exactly the same format as the fact key's value.</td>
</tr>
<tr>
<td>name</td>
<td>The node's name from the request's URL is merged into this object under this key.</td>
</tr>
</tbody>
</table>

This is an example of a valid request body:

```json
{
    "fact": {
        "ear-tips": "pointed",
        "eyebrow pitch": "40",
        "hair": "dark",
        "resting bpm": "120",
        "blood oxygen transporter": "hemocyanin",
        "anterior tricuspids": "2",
        "appendices": "1",
        "spunk": "10"
    }
}
```
Response format

The response is a JSON object that describes how the node would be classified.

- If the node would be successfully classified, this object contains the final classification.
- If the classification would fail due to conflicts, this object contains a description of the conflicts.

This response is intended to provide insight into the entire classification process, so that if a node isn't classified as expected, you can trace the deviation.

Classification proceeds in this order:

1. All node group rules are tested on the node's facts and name, and groups that don't match the node are culled, leaving the matching groups.
2. Inheritance relations are used to further cull the matching groups, by removing any matching node group that has a descendant that is also a matching node group. Those node groups that are left over are leaf groups.
3. Each leaf group is transformed into its inherited classification by adding all the inherited values from its ancestors.
4. All of the inherited classifications and individual node classifications are inspected for conflicts. A conflict occurs whenever two inherited classifications define different values for the environment, a class parameter, or a top-level variable.
5. Any individual node classification, including classes, class parameters, configuration data, and variables, is added.
6. Individual node classification is applied to the group classification, forming the final classification, which is then returned to the client.

The JSON object returned by this endpoint uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>match_explanations</td>
<td>Corresponds to step 1 of classification, finding the matching node groups. This key's value is an explanation object just like those found in node check-ins, which maps between a matching group's ID and an explained condition object that demonstrates why the node matched that group's rule.</td>
</tr>
<tr>
<td>leaf_groups</td>
<td>Corresponds to step 2 of classification, finding the leaves. This key's value is an array of the leaf groups (that is, those groups that are not related to any of the other matching groups).</td>
</tr>
<tr>
<td>inherited_classifications</td>
<td>Corresponds to step 3 of classification, adding inherited values. This key's value is an object mapping from a leaf group's ID to the classification values provided by that group (after inheritance).</td>
</tr>
<tr>
<td>conflicts</td>
<td>Corresponds to step 4 of classification. This key is present only if there are conflicts between the inherited classifications. Its value is similar to a classification object, but wherever there was a conflict there's an array of conflict details instead of a single classification value. Each of these details is an object with three keys: value, from, and defined_by. The value key is a conflicting value, the from key is the group whose inherited classification provided this value, and the defined_by key is the group that actually defined the value (which can be an ancestor of the from group).</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>individual_classification</td>
<td>Corresponds to step 5 of classification. Its value includes classes, class parameters, configuration data, and variables applied directly during this stage.</td>
</tr>
<tr>
<td>final_classification</td>
<td>Corresponds to step 6 of classification, present only if there are no conflicts between the inherited classifications. Its value is the result of merging all individual node classification and group classification.</td>
</tr>
<tr>
<td>node_as_received</td>
<td>The submitted node object as received by the service, after adding the name and, if not supplied by the client, an empty trusted object. This key does not correspond to any of the classification steps.</td>
</tr>
<tr>
<td>classification_source</td>
<td>An annotated version of the classification that has the environment and every class parameter and variable replaced with an “annotated value” object. This key does not correspond to any of the classification steps.</td>
</tr>
</tbody>
</table>

This example shows a response the endpoint could return in the case of a successful classification:

```json
{
  "node_as_received": {
    "name": "Tuvok",
    "trusted": {},
    "fact": {
      "ear-tips": "pointed",
      "eyebrow pitch": "30",
      "blood oxygen transporter": "hemocyanin",
      "anterior tricuspsids": "2",
      "hair": "dark",
      "resting bpm": "200",
      "appendices": "0",
      "spunk": "0"
    }
  },
  "match_explanations": {
    "00000000-0000-4000-8000-000000000000": {
      "value": true,
      "form": ["~", {"path": "name", "value": "Tuvok"}, ".*"]
    },
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "value": true,
      "form": ["and",
        {
          "value": true,
          "form": [">=", {"path": ["fact", "eyebrow pitch"], "value": "30"}], "25"]
        },
        {
          "value": true,
          "form": ["=", {"path": ["fact", "ear-tips"], "value": "pointed"}], "pointed"], "pointed"]
        },
        {
          "value": true,
          "form": ["=", {"path": ["fact", "hair"], "value": "dark"}], "dark"]
        }
      }
    }
  }
```
This example shows a response that resulted from conflicts:

```json
{
    "node_as_received": {
        "name": "Spock",
        "trusted": {},
        "fact": {
            "ear-tips": "pointed",
```
"eyebrow pitch": "40",
"blood oxygen transporter": "hemocyanin",
"anterior tricuspid": "2",
"hair": "dark",
"resting bpm": "120",
"appendices": "1",
"spunk": "10"
}
],
"match_explanations": {
"00000000-0000-4000-8000-000000000000": {
"value": true,
"form": ["~", {"path": "name", "value": "Spock"}, ",.*"]
},
"a130f715-c929-448b-82cd-fe21d3f83b58": {
"value": true,
"form": [">", {"path": ["fact", "spunk"], "value": "10"}, ",5"]
},
"8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
"value": true,
"form": ["and",
{
"value": true,
"form": [">"], {"path": ["fact", "eyebrow pitch"], "value": "30"}, ",25"]
},
{
"value": true,
"form": ["="], {"path": ["fact", "ear-tips"], "value": "pointed"}, ",pointed"]
},
{
"value": true,
"form": ["="], {"path": ["fact", "hair"], "value": "dark"}, ",dark"]
},
{
"value": true,
"form": [">"], {"path": ["fact", "resting bpm"], "value": "200"}, ",100"]
},
{
"value": true,
"form": ["="],
{
"path": ["fact", "blood oxygen transporter"],
"value": "hemocyanin"
},
"hemocyanin"
}
]
}],
"leaf_groups": {
"a130f715-c929-448b-82cd-fe21d3f83b58": {
"name": "Humans",
"id": "a130f715-c929-448b-82cd-fe21d3f83b58",
"parent": "00000000-0000-4000-8000-000000000000",
"rule": [">"], ["fact", "spunk"], "5"],
"environment": "alpha-quadrant",
"variables": {},
"classes": {
"emotion": {"importance": "primary"},
© 2022 Puppet, Inc., a Perforce company
"logic": {"importance": "secondary"},
"8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
"name": "Vulcans",
"id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
"parent": "00000000-0000-4000-8000-000000000000",
"rule": ["and", [">=", ["fact", "eyebrow pitch"],
"25"],
["=",["fact","ear-tips"],"pointed"],
["=",["fact","hair"],"dark"],
[">=", ["fact", "resting bpm"], "100"],
["=",["fact","blood oxygen transporter"],"hemocyanin"]
},
"environment": "alpha-quadrant",
"variables": {},
"classes": {
"emotion": {"importance": "ignored"},
"logic": {"importance": "primary"}
}
}"inherited_classifications": {
"a130f715-c929-448b-82cd-fe21d3f83b58": {
"environment": "alpha-quadrant",
"variables": {},
"classes": {
"logic": {"importance": "secondary"},
"emotion": {"importance": "primary"}
}
},
"8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
"environment": "alpha-quadrant",
"variables": {},
"classes": {
"logic": {"importance": "primary"},
"emotion": {"importance": "ignored"}
}
}"conflicts": {
"classes": {
"logic": {
"importance": [
{ "value": "secondary",
"from": {
"name": "Humans",
"id": "a130f715-c929-448b-82cd-fe21d3f83b58",
...
},
"defined_by": {
"name": "Humans",
"id": "a130f715-c929-448b-82cd-fe21d3f83b58",
...
}
},
{ "value": "primary",
"from": {
"name": "Vulcans",
"id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
...
}
Commands endpoint

Use the commands endpoint to unpin specified nodes from all groups they’re pinned to.

POST /v1/commands/unpin-from-all

Use the /v1/commands/unpin-from-all to unpin specified nodes from all groups they’re pinned to. Nodes that are dynamically classified using rules aren’t affected.

Nodes are unpinned from only those groups for which you have view and edit permissions. Because group permissions are applied hierarchically, you must have Create, edit, and delete child groups or Edit child group rules permissions for the parent groups of the groups you want to unpin the node from.

Note: As number of groups increases, response time can increase significantly with the unpin-from-all endpoint.

Request format

The request body must be a JSON object describing the nodes to unpin, using the following key:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>The certname of the nodes (required).</td>
</tr>
</tbody>
</table>

For example:

```
{"nodes": ["foo", "bar"]}
```

Response format

If unpinning is successful, the service returns a list of nodes with the groups they were unpinned from. If a node wasn’t pinned to any groups, it’s not included in the response.

```
{"nodes": [{"name": "foo",  "groups": [{"id": "8310b045-c244-4008-88d0-b49573c84d2d",  "name": "Webservers",  "environment": "production"},  {"id": "84b1b9b1-6db5-4897-9409-a4a3a94b7f09",  "name": "Test",  "environment": "test"}]},  {"name": "bar",  "groups": [{"id": "84b1b9b1-6db5-4897-9409-a4a3a94b7f09",  "name": "Test",  "environment": "test"}]}]}
```

Assuming token is set as an environment variable, the example below unpins "host1.example" and "host2.example":

```
type_header= 'Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print server):4433/classifier-api/v1/commands/unpin-from-all"
data='{"nodes": ["host1.example", "host2.example"]}''
curl --insecure --header "$type_header" --header "$auth_header"--request POST "$url" --data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

The classifier responds with information about each group it is removed from:

```
{"nodes":
```

© 2022 Puppet, Inc., a Perforce company
Environments endpoint

Use the environments endpoint to retrieve information about environments in the node classifier. The output tells you either which environments are available or whether a named environment exists. The output can be helpful when creating new node groups, which must be associated with an environment. The node classifier gets its information about environments from Puppet, so do not use this endpoint to create, update, or delete them.

**GET /v1/environments**

Use the /v1/environments endpoint to retrieve a list of all environments known to the node classifier.

**Response format**

The response is a JSON array of environment objects, using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the environment (a string).</td>
</tr>
<tr>
<td>sync_succeeded</td>
<td>Whether the environment synched successfully during the last class synchronization (a Boolean).</td>
</tr>
</tbody>
</table>

**Error responses**

No error responses specific to this request are expected.

**GET /v1/environments/<name>**

Use the /v1/environments/\<name\> endpoint to retrieve the environment with the given name. The main use of this endpoint is to check if an environment actually exists.

**Response format**

If the environment exists, the service returns a 200 response with an environment object in JSON format.

**Error responses**

If the environment with the given name cannot be found, the service returns a 404: Not Found response with an empty body.

**PUT /v1/environments/<name>**

Use the /v1/environments/\<name\> endpoint to create a new environment with the given name.

**Request format**

No further information is required in the request besides the name portion of the URL.

**Response format**

If the environment is created successfully, the service returns a 201: Created response whose body is the environment object in JSON format.

**Error responses**

No error responses specific to this operation are expected.
Nodes check-in history endpoint

Use the nodes endpoint to retrieve historical information about nodes that have checked into the node classifier.

Enable check-in storage to use this endpoint

Node check-in storage is disabled by default because it can place excessive loads on larger deployments. You must enable node check-in storage before using the check-in history endpoint. If node check-in storage is not enabled, the endpoint returns an empty array.

To enable node check-in storage, set the classifier_node_check_in_storage parameter in the puppet Enterprise::profile::console class to true.

Related information

Set configuration data on page 465

Configuration data set in the console is used for automatic parameter lookup, the same way that Hiera data is used. Console configuration data takes precedence over Hiera data, but you can combine data from both sources to configure nodes.

GET /v1/nodes

Use the /v1/nodes endpoint to retrieve a list of all nodes that have checked in with the node classifier, each with their check-in history.

Query Parameters

limit

Controls the maximum number of nodes returned. limit=10 returns 10 nodes.

offset

Specifies how many nodes to skip before the first returned node. offset=20 skips the first 20 nodes.

CAUTION: In deployments with large numbers of nodes, a large or unspecified limit might cause the console-services process to run out of memory and crash.

Response format

The response is a JSON array of node objects. Each node object contains these two keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node according to Puppet (a string).</td>
</tr>
<tr>
<td>check_ins</td>
<td>An array of check-in objects (described below).</td>
</tr>
</tbody>
</table>

Each check-in object describes a single check-in of the node. The check-in objects have the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The time of the check-in as a string in ISO 8601 format (with timezone).</td>
</tr>
<tr>
<td>explanation</td>
<td>An object mapping between IDs of groups that the node was classified into and explained condition objects that describe why the node matched this group’s rule.</td>
</tr>
<tr>
<td>transaction_uuid</td>
<td>A uuid representing a particular Puppet transaction that is submitted by Puppet at classification time. This makes it possible to identify the check-in involved in generating a specific catalog and report.</td>
</tr>
</tbody>
</table>
The explained condition objects are the node group's rule condition marked up with the node's value and the result of evaluation. Each form in the rule (that is, each array in the JSON representation of the rule condition) is replaced with an object that has two keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A Boolean that is the result of evaluating this form. At the top level, this is the result of the entire rule condition, but because each sub-condition is marked up with its value, you can use this to understand, say, which parts of an or condition were true.</td>
</tr>
<tr>
<td>form</td>
<td>The condition form, with all sub-forms as further explained condition objects.</td>
</tr>
</tbody>
</table>

Besides the condition markup, the comparison operations of the rule condition have their first argument (the fact path) replaced with an object that has both the fact path and the value that was found in the node at that path.

The following example shows the format of an explained condition.

Start with a node group with the following rule:

```json
["and", [">=" , ["fact", "pressure hulls"], "1"],
[=" , ["fact", "warp cores"], "0"],
[">=" , ["fact", "docking ports"], "10"]
```

The following node checks into the classifier:

```json
{
  "name": "Deep Space 9",
  "fact": {
    "pressure hulls": "10",
    "docking ports": "18",
    "docking pylons": "3",
    "warp cores": "0",
    "bars": "1"
  }
}
```

When the node checks in for classification, it matches the above rule, so that check-in's explanation object has an entry for the node group that the rule came from. The value of this entry is this explained condition object:

```json
{
  "value": true,
  "form": ["and", {
    "value": true,
    "form": [">=" , ["fact", "pressure hulls"], "3"],
    "1"
},
  {
    "value": true,
    "form": [=" , ["fact", "warp cores"], "0"], "0"
  },
  {
    "value": true,
    "form": [">" ["path": ["fact", "docking ports"], "value": "18"], "9"]
  }
]
```
GET /v1/nodes/<node>
Use the /v1/nodes/<node> endpoint to retrieve the check-in history for only the specified node.

Response format
The response is one node object as described above in the GET /v1/nodes documentation, for the specified node. The following example shows a node object:

```json
{
  "name": "Deep Space 9",
  "check_ins": [
    {
      "time": "2369-01-04T03:00:00Z",
      "explanation": {
        "53029cf7-2070-4539-87f5-9fc754a0f041": {
          "value": true,
          "form": [
            { "value": true,
            "form": [">", {"path": ["fact", "pressure hulls"], "value": "3"}], "value": "1"}],
            { "value": true,
            "form": ["=", {"path": ["fact", "warp cores"], "value": "0"}], "value": "0"}],
            { "value": true,
            "form": [">" {"path": ["fact", "docking ports"], "value": "18"}], "value": "9"}],
        }]
    },
    "transaction_uuid": "d3653a4a-4ebe-426e-a04d-dbebec00e97f"
}
```

Error responses
If the specified node has not checked in, the service returns a 404: Not Found response, with the usual JSON error response in its body.

Group children endpoint
Use the group children endpoint to retrieve a specified group and its descendents.

GET /v1/group-children/:id
Use the /v1/group-children/:id endpoint to retrieve a specified group and its descendents.

Request format
The request body must be a JSON object specifying a group and an optional depth indicating how many levels of descendents to return.

- depth: (optional) an integer greater than or equal to 0 that limits the depth of trees returned. Zero means return the group with no children.
For example:

```
GET /v1/group-children/00000000-0000-4000-8000-000000000000?depth=2
```

**Response format**

The response is a JSON array of group objects, using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (a string).</td>
</tr>
<tr>
<td>id</td>
<td>The node group's ID, which is a string containing a type-4 (random) UUID.</td>
</tr>
<tr>
<td>description</td>
<td>An optional key containing an arbitrary string describing the node group.</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group (which is the root of the hierarchy). Note that the root group always has the lowest-possible random UUID, 00000000-0000-4000-8000-000000000000.</td>
</tr>
<tr>
<td>rule</td>
<td>A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group. See Rule condition grammar for more information on how this condition must be structured.</td>
</tr>
<tr>
<td>classes</td>
<td>An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the node group sets for that parameter (always a string).</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleted</td>
<td>An object similar the classes object that shows which classes and class parameters set by the node group have since been deleted from Puppet. If none of the node group's classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the puppetlabs.classifier/deleted key, whose value is a Boolean indicating whether the entire class has been deleted from Puppet. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: puppetlabs.classifier/deleted, mapping to a boolean indicating whether the specific class parameter has been deleted from Puppet; and value, mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the classes object).</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).</td>
</tr>
<tr>
<td>children</td>
<td>A JSON array of the group's immediate children. Children of children are included to the optionally-specified depth.</td>
</tr>
<tr>
<td>immediate_child_count</td>
<td>The number of immediate children of the group. Child count reflects the number of children that exist in the classifier, not the number that are returned in the request, which can vary based on permissions and query parameters.</td>
</tr>
</tbody>
</table>

The following is an example response from a query of the root node group with two children, each with three children. The user has permission to view only child-1 and grandchild-5, which limits the response.

```
[
  {
    "name": "child-1",
    "id": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
    "parent": "00000000-0000-4000-8000-000000000000",
    "environment_trumps": false,
    "rule": ["and", ["=", ["fact", "foo"], "bar"], ["not", ["<", ["fact", "uptime_days"], "31"]]],
    "variables": {},
    "environment": "test",
    "classes": {},
    "children": [
      {
        "name": "grandchild-1",
        "id": "a3d976ad-51d3-4a29-af57-09990f3a2481",
        "parent": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
        "environment_trumps": false,
```
Permissions
The response returned varies based on your permissions.

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>View the specified group only</td>
<td>An array containing the group and its descendents, ending at the optional depth</td>
</tr>
<tr>
<td>View descendents of the specified group, but not the group itself</td>
<td>An array starting at the roots of every tree you have permission to view and ending at the optional depth</td>
</tr>
<tr>
<td>View neither the specified group nor its descendents</td>
<td>An empty array</td>
</tr>
</tbody>
</table>

Error responses
Responses and keys returned for group requests depend on the type of error.

malformed-uuid
If the requested ID is not a valid UUID, the service returns a 400: Bad Request response using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-uuid&quot;</td>
</tr>
<tr>
<td>details</td>
<td>The malformed UUID as received by the server.</td>
</tr>
</tbody>
</table>

**malformed-number or illegal-count**

If the value of the depth parameter is not an integer, or is a negative integer, the service returns a 400: Bad Request response using one of the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-number&quot; or &quot;illegal-count&quot;</td>
</tr>
</tbody>
</table>

**Rules endpoint**

Use the rules endpoint to translate a group’s rule condition into PuppetDB query syntax.

**POST /v1/rules/translate**

Translate a group's rule condition into PuppetDB query syntax.

**Request format**

The request's body contains a rule condition as it would appear in the rule field of a group object.

The endpoint supports an optional query parameter format, which defaults to nodes. If specified as ? format=inventory, it allows you to get the classifier rules in a compatible dot notation format instead of the standard PuppetDB AST.

**Response format**

The response is a PuppetDB query string that can be used with PuppetDB nodes endpoint in order to see which nodes would satisfy the rule condition (that is, which nodes would be classified into a group with that rule).

**Error responses**

Rules that use structured or trusted facts cannot be converted into PuppetDB queries, because PuppetDB does not yet support structured or trusted facts. If the rule cannot be translated into a PuppetDB query, the server returns a 422 Unprocessable Entity response containing the usual JSON error object. The error object has a kind of "untranslatable-rule", a msg that describes why the rule cannot be translated, and contains the received rule in details.

If the request does not contain a valid rule, the server returns a 400 Bad Request response with the usual JSON error object. If the rule was not valid JSON, the error's kind is "malformed-request", the msg states that the request's body could not be parsed as JSON, and the details contain the request's body as received by the server.

If the rule does not conform to the rule grammar, the kind key is "schema-violation", and the details key is an object with submitted, schema, and error keys which respectively describe the submitted object, the schema that object is expected to conform to, and how the submitted object failed to conform to the schema.

**Related information**

GET /v1/groups on page 530
Use the /v1/groups endpoint to retrieve a list of all node groups in the node classifier.

**Import hierarchy endpoint**

Use the import hierarchy endpoint to delete all existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.

**POST /v1/import-hierarchy**

Delete all existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.

The request's body must contain an array of node groups that form a valid and complete node group hierarchy. Valid means that the hierarchy does not contain any cycles, and complete means that every node group in the hierarchy is reachable from the root.

**Request format**

The request body must be a JSON array of node group objects as described in the groups endpoint documentation. All fields of the node group objects must be defined; no default values are supplied by the service.

Note that the output of the group collection endpoint, /v1/groups, is valid input for this endpoint.

**Response format**

If the submitted node groups form a complete and valid hierarchy, and the replacement operation is successful, a 204 No Content response with an empty body is returned.

**Error responses**

If any of the node groups in the array are malformed, a 400 Bad Request response is returned. The response contains the usual JSON error payload. The kind key is "schema-violation"; the msg key contains a short description of the problems with the malformed node groups; and the details key contains an object with three keys:

- **submitted**: an array of only the malformed node groups found in the submitted request.
- **error**: an array of structured descriptions of how the node group at the corresponding index in the submitted array failed to meet the schema.
- **schema**: the structured schema for node group objects.

If the hierarchy formed by the node groups contains a cycle, then a 422 Unprocessable Entity response is returned. The response contains the usual JSON error payload, where the kind key is "inheritance-cycle", the msg key contains the names of the node groups in the cycle, and the details key contains an array of the complete node group objects in the cycle.

If the hierarchy formed by the node groups contains node groups that are unreachable from the root, then a 422 Unprocessable Entity response is returned. The response contains the usual JSON error payload, where the kind key is "unreachable-groups", the msg lists the names of the unreachable node groups, and the details key contains an array of the unreachable node group objects.

**Related information**

Groups endpoint on page 530
The groups endpoint is used to create, read, update, and delete groups.

Node classifier errors on page 577
Familiarize yourself with error responses to make working the node classifier service API easier.

**Last class update endpoint**
Use the last class update endpoint to retrieve the time that classes were last updated from the primary server.

GET /v1/last-class-update
Use the /v1/last-class-update endpoint to retrieve the time that classes were last updated from the primary server.

**Response**
The response is always an object with one field, last_update. If there has been an update, the value of last_update field is the time of the last update in ISO8601 format. If the node classifier has never updated from Puppet, the field is null.

**Update classes endpoint**
Use update classes endpoint to trigger the node classifier to update class and environment definitions from the primary server.

POST /v1/update-classes
Use the /v1/update-classes endpoint to trigger the node classifier to update class and environment definitions from the primary server. The classifier service uses this endpoint when you refresh classes in the console.

**Note:** If you don't use Code Manager and you changed the default value of the environment-class-cache-enabled server setting, you must manually delete the environment cache before using this endpoint.

**Query parameters**
The request accepts the following optional parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>If provided, fetches classes for only the specified environment.</td>
</tr>
</tbody>
</table>

For example:

cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/update-classes?environment=production"
curl --cert "$cert" --cacert "$cacert" --key "$key" --request POST "$uri"

See Usage notes for curl examples for information about forming curl commands.

**Response**
For a successful update, the service returns a 201 response with an empty body.

**Error responses**
If the primary server returns an unexpected status to the node classifier, the service returns a 500: Server Error response with the following keys:
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;unexpected-response&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Describes the error</td>
</tr>
<tr>
<td>details</td>
<td>A JSON object, which has <code>url</code>, <code>status</code>, <code>headers</code>, and <code>body</code> keys describing the response the classifier received from the primary server</td>
</tr>
</tbody>
</table>

**Related information**

*Enable or disable cached data when updating classes* on page 289

The optional `environment-class-cache-enabled` setting specifies whether cached data is used when updating classes in the console. When `true`, Puppet Server refreshes classes using file sync, improving performance.

**Validation endpoints**

Use validation endpoints to validate groups in the node classifier.

**POST /v1/validate/group**

Use the `/v1/validate/group` endpoint to validate groups in the node classifier.

**Request format**

The request contains a group object. The request uses the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the node group (required).</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the node group's environment. This key is optional; if it's not provided, the default environment (production) is used.</td>
</tr>
<tr>
<td>environment_trumps</td>
<td>Whether this node group's environment overrides those of other node groups at classification-time. This key is optional; if it's not provided, the default value of <code>false</code> is used.</td>
</tr>
<tr>
<td>description</td>
<td>A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.</td>
</tr>
<tr>
<td>parent</td>
<td>The ID of the node group's parent (required).</td>
</tr>
<tr>
<td>rule</td>
<td>The condition that must be satisfied for a node to be classified into this node group. The structure of this condition is described in the &quot;Rule Condition Grammar&quot; section above.</td>
</tr>
<tr>
<td>variables</td>
<td>An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The <code>variables</code> key is optional, and if a node group does not define any top-level variables then it can be omitted.</td>
</tr>
</tbody>
</table>
### Key Definition

**classes**

An object that defines the classes to be used by nodes in the node group. The classes key is required, and at minimum is an empty object ({}). The classes key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The classes key is not optional; if it is missing, the service returns a 400: Bad Request response.

### Response format

If the group is valid, the service returns a 200 OK response with the validated group as the body.

If a validation error is encountered, the service returns one of the following 400-level error responses.

#### schema-violation

If any of the required keys are missing or the values of any of the defined keys do not match the required type, the service returns a 400: Bad Request response using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;schema-violation&quot;</td>
</tr>
</tbody>
</table>
| details | An object that contains three keys:  
|        |   • code>submitted</code>: Describes the submitted object.  
|        |   • schema: Describes the schema that object is expected to conform to.  
|        |   • error: Describes how the submitted object failed to conform to the schema. |

#### malformed-request

If the request's body could not be parsed as JSON, the service returns a 400: Bad Request response using the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;malformed-request&quot;</td>
</tr>
</tbody>
</table>
Key | Definition
--- | ---
details | An object that contains two keys:
  - body: Holds the request body that was received.
  - error: Describes how the submitted object failed to conform to the schema.

uniqueness-violation

If your attempt to create the node group violates uniqueness constraints (such as the constraint that each node group name must be unique within its environment), the service returns a 422: Unprocessable Entity response using the following keys:

Key | Definition
--- | ---
kind | "uniqueness-violation"
msg | Describes which fields of the node group caused the constraint to be violated, along with their values.
details | An object that contains two keys:
  - conflict: An object whose keys are the fields of the node group that violated the constraint and whose values are the corresponding field values.
  - constraintName: The name of the database constraint that was violated.

missing-referents

If classes or class parameters defined by the node group, or inherited by the node group from its parent, do not exist in the submitted node group's environment, the service returns a 422: Unprocessable Entity response. In both cases the response object uses the following keys:

Key | Definition
--- | ---
kind | "missing-referents"
msg | Describes the error and lists the missing classes or parameters.
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>details</td>
<td>An array of objects, where each object describes a single missing referent, and has the following keys:</td>
</tr>
<tr>
<td></td>
<td>• kind: &quot;missing-class&quot; or &quot;missing-parameter&quot;, depending on whether the entire class doesn't exist, or the class just doesn't have the parameter.</td>
</tr>
<tr>
<td></td>
<td>• missing: The name of the missing class or class parameter.</td>
</tr>
<tr>
<td></td>
<td>• environment: The environment that the class or parameter is missing from; that is, the environment of the node group where the error was encountered.</td>
</tr>
<tr>
<td></td>
<td>• group: The name of the node group where the error was encountered. Due to inheritance, this might not be the group where the parameter was defined.</td>
</tr>
<tr>
<td></td>
<td>• defined_by: The name of the node group that defines the class or parameter.</td>
</tr>
</tbody>
</table>

missing-parent

If the parent of the node group does not exist, the service returns a 422: Unprocessable Entity response. The response object uses the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;missing-parent&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Shows the parent UUID that did not exist.</td>
</tr>
<tr>
<td>details</td>
<td>The full submitted node group.</td>
</tr>
</tbody>
</table>

inheritance-cycle

If the request causes an inheritance cycle, the service returns a 422: Unprocessable Entity response. The response object uses the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&quot;inheritance-cycle&quot;</td>
</tr>
<tr>
<td>details</td>
<td>An array of node group objects that includes each node group involved in the cycle</td>
</tr>
<tr>
<td>msg</td>
<td>A shortened description of the cycle, including a list of the node group names with each followed by its parent until the first node group is repeated.</td>
</tr>
</tbody>
</table>

Node classifier errors

Familiarize yourself with error responses to make working the node classifier service API easier.
Error response description
Errors from the node classifier service are JSON responses.

Error responses contain these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>A string classifying the error. It is the same for all errors that have the same kind of thing in their details key.</td>
</tr>
<tr>
<td>msg</td>
<td>A human-readable message describing the error, suitable for presentation to the user.</td>
</tr>
<tr>
<td>details</td>
<td>Additional machine-readable information about the error condition. The format of this key's value varies between kinds of errors but is the same for any given error kind.</td>
</tr>
</tbody>
</table>

**Internal server errors**
Any endpoint might return a 500: Internal Server Error response in addition to its usual responses. There are two kinds of internal server error responses: application-error and database-corruption.

An application-error response is a catchall for unexpected errors. The msg of an application-error 500 contains the underlying error's message first, followed by a description of other information that can be found in details. The details contain the error's stack trace as an array of strings under the trace key, and might also contain schema, value, and error keys if the error was caused by a schema validation failure.

A database-corruption 500 response occurs when a resource that is retrieved from the database fails to conform to the schema expected of it by the application. This is probably just a bug in the software, but it could potentially indicate either genuine corruption in the database or that a third party has changed values directly in the database. The msg section contains a description of how the database corruption could have occurred. The details section contains retrieved, schema, and error keys, which have the resource as retrieved, the schema it is expected to conform to, and a description of how it fails to conform to that schema as the respective values.

**Not found errors**
Any endpoint where a resource identifier is supplied can produce a 404 Not Found Error response if a resource with that identifier could not be found.

All not found error responses have the same form. The kind is "not-found", the msg is "The resource could not be found.", and the details key contains the URI of the request that resulted in this response.

**Node classifier API v2**
These are the endpoints for the node classifier v2 API.

- Classification endpoint on page 579

The classification endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

**Related information**
API index on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Classification endpoint**

The classification endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

- **POST /v2/classified/nodes/<name>** on page 579

Use the `/v2/classified/nodes/<name>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

**POST /v2/classified/nodes/<name>**

Use the `/v2/classified/nodes/<name>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

**Request format**

The request body can contain a JSON object describing the facts and trusted facts of the node to be classified. The object can have these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact</td>
<td>The regular, non-trusted facts of the node. The value of this key is a further object, whose keys are fact names, and whose values are the fact values. Fact values can be a string, number, boolean, array, or object.</td>
</tr>
<tr>
<td>trusted</td>
<td>The trusted facts of the node. The values of this key are subject to the same restrictions as those on the value of the fact key.</td>
</tr>
</tbody>
</table>

**Response format**

The response is a JSON object describing the node post-classification, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>groups</td>
<td>An array of the groups that this node was classified into. The value of this key is an array of hashes containing the id and the name, sorted by name</td>
</tr>
<tr>
<td>environment</td>
<td>The name of the environment that this node uses, which is taken from the node groups the node was classified into.</td>
</tr>
<tr>
<td>classes</td>
<td>An array of the classes (strings) that this node received from the groups it was classified into.</td>
</tr>
<tr>
<td>parameters</td>
<td>An object describing class parameter values for the above classes wherever they differ from the default. The keys of this object are class names, and the values are further objects describing the parameters for just the associated class. The keys of that innermost object are parameter names, and the values are the parameter values, which can be any sort of JSON value.</td>
</tr>
</tbody>
</table>

This is an example of a response from this endpoint:

```json
{
  "groups": [{"id": "9c0c7d07-a199-48b7-9999-3cdf7654e0bf",...
```
Error responses

If the node is classified into multiple node groups that define conflicting classifications for the node, the service returns a 500 Server Error response.

The body of this response contains the usual JSON error object described in the errors documentation.

The kind key of the error is "classification-conflict", the msg describes generally why this happens, and the details key contains an object that describes the specific conflicts encountered.

The keys of these objects are the names of parameters that had conflicting values defined, and the values are arrays of value detail objects. The details object may have between one and all of the following three keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>Maps directly to an array of value detail objects (described below).</td>
</tr>
<tr>
<td>variables</td>
<td>Contains an object with a key for each conflicting variable, whose values are an array of value detail objects.</td>
</tr>
<tr>
<td>classes</td>
<td>Contains an object with a key for each class that had conflicting parameter definitions, whose values are further objects that describe the conflicts for that class's parameters.</td>
</tr>
</tbody>
</table>

A value details object describes one of the conflicting values defined for the environment, a variable, or a class parameter. Each object contains these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The defined value, which is a string for environment and class parameters, but for a variable can be any JSON value.</td>
</tr>
<tr>
<td>from</td>
<td>The node group that the node was classified into that caused this value to be added to the node's classification. This group cannot define the value, because it can be inherited from an ancestor of this group.</td>
</tr>
<tr>
<td>defined_by</td>
<td>The node group that actually defined this value. This is often the from group, but could instead be an ancestor of that group.</td>
</tr>
</tbody>
</table>

This example shows a classification conflict error object with node groups truncated for clarity:

```json
{
  "kind": "classification-conflict",
  "environment": "staging",
  "classes": ["apache"],
  "parameters": {
    "apache": {
      "keepalive_timeout": 30,
      "log_level": "notice"
    }
  }
}
```
"msg": "The node was classified into multiple unrelated groups that
defined conflicting class parameters or top-level variables. See `details`
for a list of the specific conflicts.",
"details": {
  "classes": {
    "songColors": {
      "blue": [
        {
          "value": "Blue Suede Shoes",
          "from": {
            "name": "Elvis Presley",
            "classes": {},
            "rule": ["=", "nodename", "the-node"],
            ...,
          },
          "defined_by": {
            "name": "Carl Perkins",
            "classes": {"songColors": {"blue": "Blue Suede Shoes"}},
            "rule": ["not", ["=", "nodename", "the-node"]],
            ...,
          }
        },
        {
          "value": "Since You've Been Gone",
          "from": {
            "name": "Aretha Franklin",
            "classes": {"songColors": {"blue": "Since You've Been Gone"}},
            ...,
          },
          "defined_by": {
            "name": "Aretha Franklin",
            "classes": {"songColors": {"blue": "Since You've Been Gone"}},
            ...,
          }
        }
      ]
    }
  }
}

In this example, the conflicting "Blue Suede Shoes" value was included in the classification because the node
matched the "Elvis Presley" group (since that is the value of the "from" key), but that group doesn't define the "Blue
Suede Shoes" value. That value is defined by the "Carl Perkins" group, which is an ancestor of the "Elvis Presley"
group, causing the latter to inherit the value from the former. The other conflicting value, "Since You've Been Gone",
is defined by the same group that the node matched.

Node inventory API

These are the endpoints for the node inventory v1 API.

Related information

API index on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Node inventory API: forming requests**

Make well-formed HTTP(S) requests to the Puppet inventory service API.

By default, the node inventory service listens on port 8143, and all endpoints are relative to the /inventory/v1 path. For example, the full URL for the /command/create-connection endpoint on localhost is https://localhost:8143/inventory/v1/command/create-connection.

**Token authentication**

All requests made to the inventory service API require authentication. You do this for each endpoint in the service using user authentication tokens contained within the X-Authentication request header.

**Example token usage: create a connection entry**

To post a new connection entry to the inventory service when running on localhost, first generate a token with the puppet-access tool.

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/command/create-connection"
data='{ "certnames": ["new.node"],
    "type": "ssh",
    "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
    },
    "sensitive_parameters": {
        "username": "root",
        "password": "password"
    },
    "duplicates": "replace"
}
curl --insecure --header "$type_header" --header "$auth_header" --request
    POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

**Example token usage: query connections for a certname**

To query the /query/connections endpoint, use the same token and header pattern.

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/query/connections?certname='new.node'"
curl --insecure --header "$type_header" --header "$auth_header" --request
    POST "$uri"
```

**Related information**

[Token-based authentication](#) on page 347
Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

POST /command/create-connection

Create a new connection entry in the inventory service database.

Connection entries contain connection options, such as credentials, which are used to connect to the certnames provided in the payload via the provided connection type.

This endpoint also inserts each of the provided certnames into PuppetDB with an empty fact set, if they are not already present. After certnames have been added to PuppetDB, you can view them from the Nodes page in the Puppet Enterprise console. You can also add them to an inventory node list when you set up a job to run tasks.

**Important:** When the Puppet orchestrator targets a certname to run a task, it first considers the value of the hostname key present in the parameters, if available. Otherwise, it uses the value of the certnames key as the hostname. A good practice is to include a hostname key only when the hostname differs from the certname. Do not include a hostname key for multiple certname connection entries.

**Request format**

The request body must be a JSON object.

- **certnames**
  - required Array containing the list of certnames to associate with connection information.

- **type**
  - required String containing either ssh or winrm. Instructs bolt-server which connection type to use to access the node when running a task.

- **parameters**
  - required Object containing arbitrary key/value pairs. The necessary parameters for connecting to the provided certnames.

  **CAUTION:** A hostname key entered here takes precedence over the values in certnames key.

- **sensitive_parameters**
  - required Object containing arbitrary key/value pairs. The necessary sensitive data for connecting to the provided certnames, stored in an encrypted format.

- **duplicates**
  - required String containing either error or replace. Instructs how to handle cases where one or more provided certnames conflict with existing certnames stored in the inventory connections database. error results in a 409 response if any certnames are duplicates. replace overwrites the existing certnames if there are conflicts.

**Request examples**

```json
{
    "certnames": ["sshnode1.example.com", "sshnode2.example.com"],
    "type": "ssh",
    "parameters": {
        "port": 1234,
        "connect-timeout": 90,
        "user": "inknowahorse",
        "run-as": "fred"
    },
    "sensitive_parameters": {
        "password": "password",
```
"sudo-password": "xtheowl",
"duplicates": "replace"
}

```cpp
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/command/create-connection"
data='{"certnames": [
  "sshnode1.example.com",
  "sshnode2.example.com"
],
  "type": "ssh",
  "parameters": {
    "port": 1234,
    "connect-timeout": 90,
    "user": "inknowahorse",
    "run-as": "fred"
  },
  "sensitive_parameters": {"password": "password",
    "sudo-password": "xtheowl"
  },
  "duplicates": "replace"
}',
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

**Response format**

If the request is valid according to the schema and the entry is successfully recorded in the database, the server returns a 201 response. The response has the same format for single and multiple certname entries.

The response is a JSON object containing the connection_id.

**connection_id**

A unique identifier that an be used to reference the record.

**Response example**

```
{
  "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec"
}
```

**SSH parameters**

To create a connection with type ssh, the following parameters and sensitive parameters are valid.

**parameters**

*Required*. Object containing the following key/value pairs and the necessary parameters for connecting to the provided certnames.

**user**

*Required* string. The user to log in as when connecting to the host.
port
   Optional integer. Connection port with the default of 22.

connect-timeout
   Optional integer. The length of time, in seconds, PE should wait when establishing connections.

run-as
   Optional string. After login, the user name to use for running commands. The default is the same value as user.

tmpdir
   Optional string. The directory to upload and execute temporary files on the target, if different than /temp.

tty
   Optional boolean. Enable text terminal allocation.

hostname
   Optional string. The hostname to connect to. Should only be provided if the desired certname for the newly created node within PE is different than the hostname of the intended target, in which case the certname field in the request body should be the desired certname, and the hostname field in the parameters object should be the hostname to connect to. As a good practice, do not provide a hostname key for connection entries with multiple certnames, and only include it if the certname is different from the hostname of the intended target. If left unspecified, the Orchestrator will connect to each host using its certname as the hostname.

sensitive_parameters
   Required. Object containing arbitrary key/value pairs and the necessary sensitive data for connecting to the provided certnames, to be stored in encrypted format.

password
   Optional string. Password to authenticate. One of either password or private-key-content is required.

private-key-content
   Optional string. Contents of a private key. One of either password or private-key-content is required.

sudo-password
   Optional string. Password to use when changing users via run-as. Should only be included if run-as is specified in the parameters object.

WinRM parameters
For creating a connection with type winrm, the following parameters and sensitive parameters are valid.

parameters
   Required. Object containing the following key/value pairs and the necessary parameters for connecting to the provided certnames.

user
   Required string. The user to log in as when connecting to the host.

port
   Optional integer. Connection port with the default of 22.

connect-timeout
   Optional integer. The length of time, in seconds, PE should wait when establishing connections.

tmpdir
   Optional string. The directory to upload and execute temporary files on the target, if different than /temp.
extensions

Optional array. List of file extensions that are accepted for tasks.

hostname

Optional string. The hostname to connect to. Should only be provided if the desired certname for the newly created node within PE is different than the hostname of the intended target, in which case the certname field in the request body should be the desired certname, and the hostname field in the parameters object should be the hostname to connect to. As a good practice, do not provide a hostname key for connection entries with multiple certnames, and only include it if the certname is different from the hostname of the intended target. If left unspecified, the Orchestrator will connect to each host using its certname as the hostname.

sensitive_parameters

Required. Object containing arbitrary key/value pairs and the necessary sensitive data for connecting to the provided certnames, to be stored in an encrypted format.

password

Required string. The password used to authenticate.

POST /command/delete-connection

Delete certnames from all associated connection entries in the inventory service database and update those certname entries in the PuppetDB with the flag preserve: false.

Request format

The request body must be a JSON object containing a certnames key.

certnames

required Array containing the list of certnames to be removed.

Request examples

```json
{
   "certnames": ["avery.gooddevice", "amediocre.device"]
}
```

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print server):8143/inventory/v1/command/delete-connection"
data='{"certnames": ["avery.gooddevice","amediocre.device"]}'
curl --insecure --header "$type_header" --header "$auth_header"--request POST "$url" --data "$data"

See Usage notes for curl examples for information about forming curl commands.

Response format

If the request matches the schema and is processed successfully, the service responds with a 204 status code and no body payload. If the user is unauthorized, the service responds with a 403 error code.

Command endpoint error responses

General format of error responses.

Every error response from the inventory service is a JSON response. Each response is an object that contains the following keys:
kind
The kind of error encountered.

msg
The message associated with the error.

details
A hash with more information about the error.

For example, if the request is missing the required content type headers, the following error occurs:

```
{
    "kind" : "puppetlabs.inventory/not-acceptable",
    "msg" : "accept must include content-type json",
    "details" : ""
}
```  

### kind error responses

For this endpoint, the `kind` key of the error displays the conflict.

- **puppetlabs.inventory/unknown-error**
  - If an unknown error occurs during the request, the server returns a 500 response.

- **puppetlabs.inventory/not-acceptable**
  - If the content provided to the API contains an "accepts" header which does not allow for JSON, the server returns a 406 response.

- **puppetlabs.inventory/unsupported-type**
  - If the content provided to the API contains a "content-type" header other than JSON, the server returns a 416 response.

- **puppetlabs.inventory/json-parse-error**
  - If there is an error while processing the request body, the server returns a 400 response.

- **puppetlabs.inventory/schema-validation-error**
  - If there is a violation of the required format for the request body, the server returns a 400 response.

- **puppetlabs.inventory/not-permitted**
  - If the user requesting an action does not have the necessary permissions to do so, the server returns a 403 response.

- **puppetlabs.inventory/duplicate-certnames**
  - If the `duplicates` parameter is not set, or is set as `error`, and one or more of the certnames in the request body already exist within the inventory, the server returns a 409 response.

### GET /query/connections

List all the connections entries in the inventory database.

**Request format**

The request body must be a JSON object.

- **certname**
  - *optional* String that represents the single certname to retrieve.

- **sensitive**
  - *optional* String or boolean that instructs whether to return sensitive parameters for each connection in the response. This parameter is gated by permission validation.
extract

optional Array of keys to return for each certname. connection_id is always returned, regardless of whether it is included. When extract is not present in the body, all keys are returned.

Tip: In order to return sensitive parameters in the extract list, the sensitive query parameter must be present and resolve to true. Otherwise, they are excluded.

Response format

The response is a JSON object containing the known connections. The following keys are used:

items
Contains an array of all the known connections matching the specified (or not specified) filtering criteria. Each item under items is an object with the following keys:

connection_id
String that is the unique identifier for the connections entry.

certnames
Array of strings that contains the certnames of the matching connections entries.

type
String that describes the type of connection for the given information. For example, ssh or winrm.

parameters
Object containing arbitrary key/value pairs and describes connection options for the entry.

sensitive_parameters
when specified and permitted An object that contains arbitrary key/value pairs and describes the sensitive connection options for the entry.

Response examples

A response for a request to list all the connections entries in the inventory database: GET /query/connections

```json
{
    "items": [
        {
            "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
            "certnames": ["devices.arecool", "dont.youthink"],
            "type": "ssh",
            "parameters": {
                "tmpdir": "/tmp",
                "port": 1234
            }
        },
        {
            "connection_id": "4932bfe7-69c4-412f-b15c-ac0a7c2883f1",
            "certnames": ["managing.devices", "is.evencooler"],
            "type": "winrm",
            "parameters": {
                "tmpdir": "/tmp",
                "port": 1234
            }
        }
    ]
}
```
A response for a request to list a specific certname: GET /query/connections?certname="averygood.device"

```json
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```

A response for request to list a specific certname and its sensitive parameters: GET /query/connections?certname="averygood.device"&sensitive=true example

```json
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      },
      "sensitive_parameters": {
        "username": "johnjohnjimmyjohn",
        "password": "C7b0$ls8ltO"
      }
    }
  ]
}
```

A response for request to list a specific certname and its value for the `type` key: GET /query/connections?certname="averygood.device"&extract=["type"]

```json
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh"
    }
  ]
}
```

**POST /query/connections**

Retrieve inventory connections in bulk via certname.

**Request format**

The request body must be a JSON object.
**certnames**
Array containing the list of certnames to retrieve from the inventory database. If this key is not included, all connections are returned.

**extract**
Array of keys to return for each certname. `connection_id` is always returned, regardless of whether it is included. When `extract` is not present in the body, all keys are returned.

**Tip:** In order to return sensitive parameters in the extract list, the `sensitive` query parameter must be present and resolve to true. Otherwise, they are excluded.

**sensitive**
`optional` String or boolean that instructs whether to return sensitive parameters for each connection in the response. This parameter is gated by permission validation.

**Response format**
The response is a JSON object containing the known connections. The following keys are used:

**items**
Contains an array of all the known connections matching the specified (or not specified) filtering criteria. Each item under `items` is an object with the following keys:

**connection_id**
String that is the unique identifier for the connections entry.

**certnames**
Array of strings that contains the certnames of the matching connections entries.

**type**
String that describes the type of connection for the given information. For example, `ssh` or `winrm`.

**parameters**
Object containing arbitrary key/value pairs and describes connection options for the entry.

**sensitive_parameters**
`when specified and permitted` An object that contains arbitrary key/value pairs and describes the sensitive connection options for the entry.

**Request and response examples**
An empty request body.

```
{}
```

A response for an empty request.

```
{
    "items": [
        {
            "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
            "certnames": ["devices.arecool", "dont.youthink"],
            "type": "winrm",
            "parameters": {
                "tmpdir": "/tmp",
                "port": 1234
            }
        },
        {
            "connection_id": "4932bfe7-69c4-412f-b15c-ac0a7c2883f1",
            "certnames": ["another.certname"]
        }
    ]
}
```
"certnames": ["managing.devices", "is.evencooler"],
"type": "ssh",
"parameters": {
  "tmpdir": "/tmp",
  "port": 1234
}
]
}

A certnames request body.

{
  "certnames": ["averygood.device"]
}

A response for a certnames request.

{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}

A request to list a specific certname and its sensitive parameters: /query/connections? sensitive=true

{
  "certnames": ["averygood.device"],
  "extract": ["certnames", "sensitive_parameters"]
}

A response for a request for a specific certname and its sensitive parameters.

{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "sensitive_parameters": {
        "username": "johnjohnjimmyjohn",
        "password": "C7b0$ls8ltO"
      }
    }
  ]
}

**Query endpoint error responses**

General format of error responses.

Every error response from the inventory service is a JSON response. Each response is an object that contains the following keys:
kind
The kind of error encountered.

msg
The message associated with the error.

details
A hash with more information about the error.

For example, if the request is missing the required content type headers, the following error occurs:

```
{
    "kind" : "puppetlabs.inventory/not-acceptable",
    "msg" : "accept must include content-type json",
    "details" : ""
}
```

kind error responses
For this endpoint, the kind key of the error displays the conflict.

puppetlabs.inventory/unknown-error
If an unknown error occurs during the request, the server returns a 500 response.

puppetlabs.inventory/not-acceptable
If the content provided to the API contains an "accepts" header which does not allow for JSON, the server returns a 406 response.

puppetlabs.inventory/unsupported-type
If the content provided to the API contains a "content-type" header other than JSON, the server returns a 416 response.

puppetlabs.inventory/json-parse-error
If there is an error while processing the request body, the server returns a 400 response.

puppetlabs.inventory/schema-validation-error
If there is a violation of the required format for the request body, the server returns a 400 response.

Managing patches

Use Puppet Enterprise to configure patching node groups to meet your needs, view available operating system patches for your nodes in the console, and apply patches using the pe_patch::patch_server task.
• **Configuring patch management** on page 593
To enable patch management, create a node group for nodes you want to patch and add the node group to the **PE Patch Management** parent node group.

• **Patching nodes** on page 599
After configuring patch management, you can start applying patches to nodes. The `patch_server` task enables simply applying patches, while the `group_patching` plan performs health checks before and after patches are applied.

### Configuring patch management
To enable patch management, create a node group for nodes you want to patch and add the node group to the **PE Patch Management** parent node group.

### Patch management OS compatibility
Patch management is compatible with current Linux operating systems using YUM, APT, and Zypper package management, as well as Microsoft Windows operating systems. We currently test against the following platforms, and these are confirmed to be compatible.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlmaLinux</td>
<td>8</td>
</tr>
<tr>
<td>Amazon Linux</td>
<td>2</td>
</tr>
<tr>
<td>CentOS</td>
<td>7</td>
</tr>
<tr>
<td>Debian</td>
<td>9, 10, 11</td>
</tr>
<tr>
<td>Fedora</td>
<td>34</td>
</tr>
<tr>
<td><strong>Note:</strong> You must install cron to run patch management on Fedora. To install cron, run <code>dnf install cronie</code></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>10, 11</td>
</tr>
<tr>
<td><strong>Note:</strong> You must use Powershell 3.0 or higher to patch Windows nodes.</td>
<td></td>
</tr>
<tr>
<td>Oracle Linux</td>
<td>7, 8</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>7, 8</td>
</tr>
<tr>
<td>Rocky Linux</td>
<td>8</td>
</tr>
<tr>
<td>Scientific Linux</td>
<td>7</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>12, 15</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>18.04, 20.04, 22.04</td>
</tr>
</tbody>
</table>

**Note:** If your operating system does not support TLSv1.2 or higher, you must Enable TLSv1 or 1.1.
Where patch information comes from

Your package management software is responsible for ensuring PE can find the latest patch information available. The pe_patch module uses OS level tools or APIs to find patches for nodes. You still have to manage the configuration of your package manager, like YUM, APT, Zypper, WSUS, or Windows Update, so your nodes can search for updates. For example, if you need to go through a proxy and you use YUM, you must configure this on your own.

Patching involves two distinct steps. First, a cron job scans for new patches and uploads related details to PuppetDB as part of the pe_patch fact. You can specify when to run the cron job with parameters in the pe_patch class. Then, patches are applied to specified nodes using the pe_patch::patch_server task or the pe_patch::group_patching plan.

Note: If you need to restrict which packages/patches your OS finds and which patches are applied:

- For *nix agents patching: Pin a package using yum versionlock, apt-mark, or zypper addlock. The pinned_packages field in the pe_patch fact refers to versions locked using these methods. This is different from apt-pinning packages, which is used to prioritize packages rather than locking them at a specific version.
- For Windows agents patching: If you use WSUS or Windows Update to deliver updates, use WSUS to approve desired updates independently.

Security updates

To find security updates, the pe_patch module uses security metadata when it is available. For example, Red Hat provides security metadata as additional metadata in YUM, Debian performs checks on the repo the updates are coming from, and Windows provides this information by default.

In the console, on the Patches page, security metadata feeds into the Apply patches table where you can filter for Security updates only.

Configure Windows Update

If you are using Windows Update, we recommend you use the puppetlabs/wsus_client module and configure these parameters in the wsus_client class.

- Set the server_url parameter to the URL of your WSUS server.
- Set the auto_update_options parameter to AutoNotify to automatically download updates and notify users.

Create a node group for nodes under patch management

Create a node group for nodes you want to patch in PE and add nodes to it. For example, create a node group for testing Windows and *nix patches prior to rolling out patches to other node groups. The PE Patch Management parent node group has the pe_patch class assigned to it and is in the console by default.

Note: Adding PE infrastructure nodes to patch management node groups can cause service interruptions when certain patches are applied.

1. In the console, click Node groups, and click Add group.
2. Specify options for the new node group, then click Add.
   - Parent name – Select PE Patch Management.
   - Group name – Enter a name that describes the role of the node group, for example, patch test.
   - Environment – Select production.
   - Environment group – Do not select this option.
3. Select the patching node group you created.
4. On the **Node group details** page, on the **Rules** tab, add nodes to the group by either pinning them individually or adding a rule to automatically add nodes that meet your specifications.

| CAUTION: Do not include the same node in multiple node groups under patch management. This might cause classification conflicts. |

5. Select **Run > Puppet** in the top right corner of the page.

PE can now manage patches for the nodes in your new node group. Repeat these steps to add any additional node groups you want under patch management.

**Related information**
- **Add nodes to a node group** on page 462
- There are two ways to add nodes to a node group.

**Specify patching parameters**

Set parameters for node groups under patch management by first applying the **pe_patch** class to them, then specifying your desired parameters.

**Before you begin**

Create at least one node group under patch management.

1. On the **Node groups** page, select the patching node group you want to add parameters to.
2. If it doesn't already exist, add the **pe_patch** class to the node group.
   a) On the **Classes** tab, enter **pe_patch** and select **Add class**.
   b) Commit changes.
3. In the **pe_patch** class, add the **patch_group** parameter and specify a value that describes the nodes in this node group.

   | Tip: The **patch_group** parameter is used to identify which nodes to run patching plans against. You might specify **patch_group** names that match your node groups, or apply the same **patch_group** parameter across several patching node groups that have similar characteristics. |

4. Specify any additional patching parameters in the **pe_patch** class.
5. Commit changes.

Run Puppet on nodes in the node group before running patching tasks or plans.

**Assign a patch management blackout window**

Apply a blackout window to prevent PE from applying patches to nodes for a specified duration of time. For example, limit applying patches during an end-of-year change freeze.

**Before you begin**

Assign the **pe_patch** class to the applicable node group. See **Specify patching parameters** on page 595 for more information.

1. On the **Node groups** page, select the patching node group you want to assign a blackout window to.
2. On the **Classes** tab, under **Parameter**, add the **blackout_windows** parameter to the **pe_patch** class.
3. In the **Value** field, enter your blackout window as a JSON hash of keys and an ISO compliant timestamp.

   For example, an end of year blackout window from the beginning of the day on 15 December 2020 to the end of the day on 15 January 2021 looks like this:

   ```json
   {
     "End of year change freeze": {
       "start": "2020-12-15T00:00:00+10:00",
       "end": "2021-01-15T23:59:59+10:00"
     }
   }
   ``

4. Commit changes.

   When a user tries to patch nodes during the blackout window, the **Patch blocked** field on the **Apply patches** table changes from **No** to **Yes** for affected patches. If the user proceeds with patching, the patching task fails.

**Patch management parameters**

Configure and tune patch management by adjusting parameters in the **pe_patch** class.

- **patch_data_owner**
  - User name for the owner of the patch data. String.
  - Default: root

- **patch_data_group**
  - Group name for the owner of the patch data. String.
  - Default: root

- **patch_cron_user**
  - User account for running the cron job that scans for new patches in the background. String.
  - Default: $patch_data_owner

- **manage_yum_utils**
  - Determines if the yum_utils package should be managed by this module on RedHat family nodes. If **true**, use the yum_utils parameter to determine how it should be managed. Boolean.
  - Default: false

- **yum_utils**
  - If managed, determines what the package is set to. Enum[installed, absent, purged, held, latest]
  - Default: installed

- **block_patching_on_warnings**
  - Determines if the patching task should run if there were warnings present on the **pe_patch** fact. If **true**, the run will abort and take no action. If **false**, the run will continue and attempt to patch. Boolean.
  - Default: false

- **fact_upload**
  - Determines if puppet fact upload runs after any changes are made to the fact cache files. Boolean.
  - Default: true

- **apt_autoremove**
  - Determines if apt-get autoremove runs during reboot. Boolean.
  - Default: false

- **manage_delta_rpm**
  - Determines if the delta_rpm package should be managed by this module on RedHat family nodes. If **true**, use the delta_rpm parameter to determine how it should be managed. Boolean.
Default: false

delta_rpm
If managed, determines what the delta_rpm package is set to. Enum[installed, absent, purged, held, latest]
Default: installed

manage yum_plugin_security
Determines if the yum_plugin_security package should be managed by this module on RedHat family nodes. If true, use the yum_plugin_security parameter to determine how it should be managed. Boolean.
Default: false

yum_plugin_security
If managed, determines what the yum_plugin_security package is set to. Enum[installed, absent, purged, held, latest]
Default: installed

reboot_override
Determines if a node reboots after patching. This overrides the setting in the task. Variant, Boolean, Enum[always, never, patched, smart, default]
• always - The node always reboots during the task run, even if no patches are required.
• never (or false) - The node never reboots during the task run, even if patches are applied.
• patched (or true) - The node reboots if patches are applied.
• smart - Use the OS supplied tools, like needs_restarting on RHEL or a pending reboot check on Windows, to determine if a reboot is required, if it is reboots, or if it does not reboot.
• default - Uses whatever option is set in the reboot parameter for the pe_patch::patch_server task.
Default: default

patch_group
Identifies nodes in or across patching node groups to run patching plans against.
Default: undef

pre_patching_scriptpath
The full path to an executable script or binary on the target node to be run before patching.
Default: undef

post_patching_scriptpath
The full path to an executable script or binary on the target node to be run after patching.
Default: undef

patch_cron_hour
The hour or hours to run the cron job that scans for new patches.
Default: absent, or *

patch_cron_month
The month or months to run the cron job that scans for new patches.
Default: absent, or *

patch_cron_monthday
The monthday or monthdays to run the cron job that scans for new patches.
Default: absent, or *
**patch_cron_weekday**

The weekday or weekdays to run the cron job that scans for new patches.

Default: absent, or *

**patch_cron_min**

The min or mins to run the cron job that scans for new patches.

Default: fqdn_rand(59) - a random number between 0 and 59.

**ensure**

Use present to install scripts, cronjobs, files, etc. Use absent to clean up system that previously hosted.

Default: present

**blackout_windows**

Determines a window of time when nodes cannot be patched. Hash.

: **title** - Name of the blackout window. String.
: **start** - Start of the blackout window (ISO8601 format). String.
: **end** - End of the blackout window (ISO8601 format). String.

Default: undef

**windows_update_criteria**

Determines which types of updates Windows Update searches for. To search both software and driver updates, remove the Type argument. String.

Default: IsInstalled=0 and IsHidden=0 and Type='Software'

**Note:** See the Microsoft documentation for more information about formatting strings for Windows Update.

### Disable patch management

Use the console to disable patch management by editing the ensure parameter in the PE Patch Management node group. You can also remove patch management by deleting patching node groups.

1. In the console, click Node groups and select the PE Patch Management node group.
2. On the Classes tab, under the pe_patch class, select the ensure parameter, and change the value to absent.
3. Click Add to node group and commit the change.
4. Run Puppet.
   The client components of the pe_patch class, like cron and scripts, are removed from PE.
5. Optional: To remove patch management from your infrastructure, click Remove node group on the Node details page for the PE Patch Management node group.

**Note:** If you have any child node groups under patch management, you must remove those node groups prior to removing the PE Patch Management parent node group.

The Patch Management section in the console sidebar remains active after disabling patch management, but the Patches page no longer reports patch information.
Patching nodes

After configuring patch management, you can start applying patches to nodes. The patch_server task enables simply applying patches, while the group_patching plan performs health checks before and after patches are applied.

Patch nodes

Use the patch_server task to apply patches to nodes. You can limit patches to security or non-security updates, Windows or *nix nodes, or a specific patch group.

Before you begin

Ensure you have permission to run the pe_patch::patch_server task.

1. On the Patches page, in the Apply patches section, use the filters to specify which patches to apply to which nodes.

   Note: Filters use and logic. This means that if you select Security updates and Windows, the results include security patches for Windows nodes, not all security patches and all Windows patches.

2. Select Run > Task.

   The Run a task page appears with patching information pre-filled for the pe_patch::patch_server task.

3. Optional: In the Job details field, provide a description of the task run. This text appears on the Tasks page.

4. Optional: Under Task parameters, add optional parameters to the task. See Patching task parameters on page 599 for a full list of available parameters.

   Note: You must click Add parameter for each optional parameter-value pair you add to the task.

5. Optional: If you want to schedule the task to run later, under Schedule, select Later and choose a time.

6. Select Run task to apply patches.

To check the status of the task, look for it on the Tasks page. You can filter the results to view only pe_patch tasks.

Note: When using patch management to update core packages that affect the networking stack, the task run might look like it failed due to the PXP agent on the node losing connection with the primary server. However, the task still completes successfully. You can confirm by checking the pe_patch fact to verify the relevant packages were updated.

Patching task parameters

The pe_patch::patch_server task applies patches to nodes. When you patch nodes in the console, most of the information for the patch_server task is prefilled on the Run a task page, but you can add additional parameters to the task before you run it.

timeout

Indicates how much time elapses before the task run times out.

Accepted values: Any positive integer, in seconds.

Default: 3600

security_only

Indicates whether to apply only security patches.

Accepted values: true, false

Default: false
**yum_params**  
Indicates additional parameters to include in YUM commands, such as including or excluding repositories.  
Accepted values: String  
Default: undef

**dpkg_params**  
Indicates additional parameters to include in `apt-get` commands.  
Accepted values: String  
Default: undef

**zypper_params**  
Indicates additional parameters to include in Zypper commands.  
Accepted values: String  
Default: undef

**clean_cache**  
Indicates if YUM or dpkg caches are cleaned at the start of the task.  
Accepted values: `true`, `false`  
Default: false

**reboot**  
Indicates if and when nodes reboot during the task run.  

| Note: | If the node group you're patching has a `reboot_override` value specified, that value overrides any `reboot` parameter you specify in task runs. |

Accepted values:

- **always** — The node always reboots during the task run, even if no patches are required.
- **never** (or `false`) — The node never reboots during the task run, even if patches are applied.
- **patched** (or `true`) — The node reboots if patches are applied.
- **smart** — Use the OS supplied tools, like `needs_restarting` on RHEL or a pending reboot check on Windows, to determine if a reboot is required.

Default: never

---

### Patch nodes with built-in health checks

Use the `group_patching` plan to patch nodes with pre- and post-patching health checks. The plan verifies that Puppet is configured and running correctly on target nodes, patches the nodes, waits for any reboots, and then runs Puppet on the nodes to verify that they're still operational.

**Before you begin**

Ensure you have permission to run the `pe_patch::group_patching` plan.

1. In the console, in the **Orchestration** section, select **Plans** and then click **Run a plan**.
2. Specify plan details:
   - **Code environment** — Select the environment where you installed the module containing the plan you want to run. For example, `production`.
   - **Job description** — Provide an optional description of the plan run.
   - **Plan** — Select `pe_patch::group_patching`.
3. In the **Plan parameters** section, specify the **patch_group** that you want to base running the plan on, and optionally add other patching plan parameters.

   **Note:** The **patch_group** parameter is defined in the **pe_patch** class for node groups under patch management. For details, see Specify patching parameters on page 595.

4. Optional: In the **Schedule** section, specify if you want the plan to run at a later date and time.

5. Click **Run job**.

To check the status of the plan, look for it on the **Plans** page.

**Note:** When using patch management to update core packages that affect the networking stack, the task run might look like it failed due to the PXP agent on the node losing connection with the primary server. However, the task still completes successfully. You can confirm by checking the **pe_patch** fact to verify the relevant packages were updated.

**Patching plan parameters**
The **pe_patch::group_patching** plan verifies that Puppet is configured and running correctly on target nodes, patches the nodes, waits for any reboots, and then runs Puppet on the nodes to verify that they're still operational.

By default, the plan includes a health check which considers "healthy" any nodes on which:

- The Puppet service is enabled and running
- Noop mode and cached catalogs are not enabled
- The run interval is 30 minutes

You can modify plan behavior with several types of optional parameters:

- Patching options let you control how patching itself is applied, including adding an optional string to arguments passed to your package provider.
- Health check options control when a pre-patching health check and a post-patching Puppet run occurs.
  
  **Tip:** The **health_check_*** parameters apply patches only to nodes that match values you specify. For example, if you change **health_check_service_running** to **false**, the pre-patching health check marks nodes on which the Puppet service is running as "unhealthy" and skips patching them.

- Reboot options control when a post-patching reboot occurs, and let you specify a script to execute after patching.

**Patching options**

**patch_group**

Specifies the **patch_group**, as defined in the **pe_patch** class parameter, that you want to base running the plan on.

Accepted values: String

**patch_task_timeout**

Indicates how much time elapses before the task run times out.

Accepted values: Any positive integer, in seconds.

Default: 3600

**security_only**

Indicates whether to apply only security patches.

Accepted values: **true, false**

Default: **false**

**yum_params**

Indicates additional parameters to include in YUM commands, such as including or excluding repositories.

Accepted values: String
Default: undef

dpkg_params
Indicates additional parameters to include in `apt-get` commands.
Accepted values: String
Default: undef

zypper_params
Indicates additional parameters to include in Zypper commands.
Accepted values: String
Default: undef

clean_cache
Indicates if YUM or dpkg caches are cleaned at the start of the task.
Accepted values: true, false
Default: false

sequential_patching
Indicates if nodes in the specified patch group are patched, rebooted, and the post-reboot script run one at a time rather than all at once.
Accepted values: true, false
Default: false

Health check options

run_health_check
Indicates whether to do a pre-patching health check and a post-patching Puppet run.
Accepted values: true, false
Default: true

health_check_noop
Verifies the `noop` setting during pre-patching health checks.
Accepted values: true, false
Default: false

health_check_runinterval
Verifies the `runinterval` setting during pre-patching health checks.
Accepted values: Any positive integer, in seconds.
Default: 1800 (equivalent to the default Puppet run interval of 30 minutes)

health_check_service_running
Verifies whether the Puppet service is running during pre-patching health checks.
Accepted values: true, false
Default: true

health_check_service_enabled
Verifies whether the Puppet service is enabled during pre-patching health checks.
Accepted values: true, false
Default: true

health_check_use_cached_catalog
Verifies the `use_cached_catalog` setting during pre-patching health checks.
Accepted values: true, false
Default: false

Reboot options

reboot
Indicates if and when nodes reboot during the plan run.

**Note:** If the node group you're patching has a reboot_override value specified, that value overrides any reboot parameter you specify in plan runs.

Accepted values:
- always — The node always reboots during the plan run, even if no patches are required.
- never — The node never reboots during the plan run, even if patches are applied.
- patched — The node reboots if patches are applied.
- smart — Use the OS supplied tools, like needs_restarting on RHEL or a pending reboot check on Windows, to determine if a reboot is required.

Default: patched

reboot_wait_time
Indicates how long to wait for nodes to reboot before running a post-patching health check.

Accepted values: Any positive integer, in seconds.
Default: 600

post_reboot_scriptpath
The full path to an executable script or binary on the target node to be run after reboot and before the final Puppet run.

Accepted values: File path
Default: undef

Orchestrating Puppet runs, tasks, and plans

Puppet orchestrator is an effective tool for making on-demand changes to your infrastructure.

With orchestrator you can initiate Puppet, task, or plan runs whenever you need them, eliminating manual work across your infrastructure.
• How Puppet orchestrator works on page 604
With the Puppet orchestrator, you can run Puppet, tasks, or plans on-demand.

• Setting up the orchestrator workflow on page 607
The orchestrator—used alongside other Puppet Enterprise (PE) tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

• Configuring Puppet orchestrator on page 614
Once you’ve installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

• Running Puppet on demand on page 620
The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

• Tasks in PE on page 633
Tasks are ad-hoc actions you can execute on a target and run from the command line or the console.

• Plans in PE on page 663
Plans allow you to tie together tasks, scripts, commands, and other plans to create complex workflows with refined access control. You can install modules that contain plans or write your own, then run them from the console or the command line.

• Orchestrator API v1 endpoints on page 695
Use this API to gather details about the orchestrator jobs you run.

• Migrating Bolt tasks and plans to PE on page 760
If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a Puppet Enterprise (PE) environment. This lets you manage and run tasks and plans using PE and the console. Bolt projects have the same structure as Puppet modules, and they can be loaded from the modules directory of a PE environment.

How Puppet orchestrator works

With the Puppet orchestrator, you can run Puppet, tasks, or plans on-demand.

When you run Puppet on-demand with the orchestrator, you control the rollout of configuration changes when and how you want them. You control when Puppet runs and where node catalogs are applied (from the environment level to an individual node). You no longer need to wait on arbitrary run times to update your nodes.

Puppet tasks allow you to execute actions on target machines. A "task" is a single action that you execute on the target via an executable file. For example, do you want to upgrade a package or restart a particular service? Set up a Puppet task run to enforce to make those changes at will.

Puppet plans are bundles of tasks that can be combined with other logic. They allow you to do complex operations, like run multiple tasks with one command or automatically run certain tasks based on the output of another task.

Tasks and plans are packaged and distributed as Puppet modules.

Puppet orchestrator technical overview

The orchestrator uses pe-orchestration-services, a JVM-based service in PE, to execute on-demand Puppet runs on agent nodes in your infrastructure. The orchestrator uses PXP agents to orchestrate changes across your infrastructure.

The orchestrator (as part of pe-orchestration-services) controls the functionality for the puppet job, puppet task, and puppet plan commands, and also controls the functionality for jobs and single node runs in the PE console.

The orchestrator is comprised of several components, each with their own configuration and log locations.

Puppet orchestrator architecture

The functionality of the orchestrator is derived from the Puppet Execution Protocol (PXP), the Puppet Communications Protocol (PCP), and the Agentless Catalog Executor (ACE) Server.
• PXP: A message format used to request that a task be executed on a remote host and receive responses on the status of that task. This is used by the pe-orchestration services to run Puppet on agents.

• PXP agent: A system service in the agent package that runs PXP.

• PCP: The underlying communication protocol that describes how PXP messages get routed to an agent and back to the orchestrator.

• PCP broker: A JVM-based service that runs in pe-orchestration-services on the primary server and in the pe-puppetserver service on compilers. PCP brokers route PCP messages, which declare the content of the message via message type, and identify the sender and intended recipient. PCP brokers on compilers connect to the orchestrator, and the orchestrator uses the brokers to direct messages to PXP agents connected to the compilers. When using compilers, PXP agents running on PE components (the primary server, PuppetDB, and the PE console) connect directly to the orchestrator, but all other PXP agents connect to compilers via load balancers.

• ACE service: A Ruby service that enables you to execute tasks, plans, and Puppet runs on remotely on agentless targets. See The ACE server configuration docs for more information.

What happens during an on-demand run from the orchestrator?

Several PE services interact when you run Puppet on demand from the orchestrator.

1. You use the puppet-job command to create a job in orchestrator.
2. The orchestrator validates your token with the PE RBAC service.
3. The orchestrator requests environment classification from the node classifier for the nodes targeted in the job, and it queries PuppetDB for the nodes.
4. The orchestrator creates the job ID and starts polling nodes in the job to check their statuses.
5. The orchestrator queries PuppetDB for the agent version on the nodes targeted in the job.
6. The orchestrator tells the PCP broker to start runs on the nodes targeted in the job, and Puppet runs start on those agents.
7. The agent sends its run results to the PCP broker.
8. The orchestrator receives run results, and requests the node run reports from PuppetDB.

What happens during a task run from the orchestrator?

Several services interact for a task run as well. Because tasks are Puppet code, they must be deployed into an environment on the primary server. Puppet Server then exposes the task metadata to the orchestrator. When a task is run, the orchestrator sends the PXP agent a URL of where to fetch the task from the primary server and the checksum of the task file. The PXP agent downloads the task file from the URL and caches it for future use. The file is validated against the checksum before every execution. The following are the steps in this process.

1. The PE client sends a task command.
2. The orchestrator checks if a user is authorized.
3. The orchestrator fetches the node target from PuppetDB if the target is a query, and returns the nodes.
4. The orchestrator requests task data from Puppet Server.
5. Puppet Server returns task metadata, file URIs, and file SHAs.
6. The orchestrator validates the task command and then sends the job ID back to the client.
7. The orchestrator sends task parameters and file information to the PXP agent.
8. The PXP agent sends a provisional response to the orchestrator, checks the SHA against the local cache, and requests the task file from Puppet Server.
9. Puppet Server returns the task file to the PXP agent.
10. The task runs.
11. The PXP agent sends the result to the orchestrator.
12. The client requests events from the orchestrator.
13. The orchestrator returns the result to the client.
**Configuration notes for the orchestrator and related components**

Configuration and tuning for the components in the orchestrator happens in various files.

- **pe-orchestration-services**: The underlying service for the orchestrator. The main configuration file is `/etc/puppetlabs/orchestration-services/conf.d`.
  
  Additional configuration for large infrastructures can include tuning the pe-orchestration-services JVM heap size, increasing the limit on open file descriptors for pe-orchestration-services, and tuning ARP tables.

- **PCP broker**: Part of the pe-puppetserver service. The main configuration file is `/etc/puppetlabs/puppetserver/conf.d`.
  
  The PCP broker requires JVM memory and file descriptors, and these resources scale linearly with the number of active connections. Specifically, the PCP broker requires:
  
  - Approximately 40 KB of memory (when restricted with the `-Xmx` JVM option)
  - One file descriptor per connection
  - An approximate baseline of 60 MB of memory and 200 file descriptors
  
  For a deployment of 100 agents, expect to configure the JVM with at least `-Xmx64m` and 300 file descriptors. Message handling requires minimal additional memory.

- **PXP agent**: Configuration is managed by the agent profile (`puppet_enterprise::profile::agent`).
  
  The PXP agent is configured to use Puppet’s SSL certificates and point to one PCP broker endpoint. If disaster recovery is configured, the agent points to additional PCP broker endpoints in the case of failover.

**Note**: If you reuse an existing agent with a new orchestrator instance, you must delete the pxp-agent spool directory, located at `/opt/puppetlabs/pxp-agent/spool` (*nix) or `C:\ProgramData\PuppetLabs\pxp-agent\var\spool` (Windows)

**Related information**

- Configure the orchestrator and pe-orchestration-services on page 303

**Debugging the orchestrator and related components**

If you need to debug the orchestrator or any of its related components, the following log locations might be helpful.

- **pe-orchestration-services**: The main log file is `/var/log/puppetlabs/orchestration-services/orchestration-services.log`.

- **PCP**: The main log file for PCP brokers on compilers is `/var/log/puppetlabs/puppetserver/pcp-broker.log`. You can configure logback through the Puppet server configuration.

  The main log file for PCP brokers on the primary server is `/var/log/puppetlabs/orchestration-services/pcp-broker.log`.

  You can also enable an access log for messages.

- **PXP agent**: The main log file is `/var/log/puppetlabs/pxp-agent/pxp-agent.log` (on *nix) or `C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log` (on Windows). You can configure this location as necessary.

  Additionally, metadata about Puppet runs triggered via the PXP agent are kept in the spool-dir, which defaults to `/opt/puppetlabs/pxp-agent/spool` (on *nix) and `C:/ProgramData/PuppetLabs/pxp-agent/var/spool` (on Windows). Results are kept for 14 days.

- **ACE server**: The main log file is `/var/log/puppetlabs/ace-server/ace-server.log`. 

© 2022 Puppet, Inc., a Perforce company
Setting up the orchestrator workflow

The orchestrator—used alongside other Puppet Enterprise (PE) tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

This recommended workflow gives you precise control over rolling out changes, such as deploying new Puppet code or updating data and classifying nodes. In this workflow, you configure your agents to use cached catalogs during scheduled runs, and you use orchestrator jobs to send new catalogs only when you're ready. Scheduled runs continue to enforce the desired state from the last orchestration job until you send another new catalog.

Before you begin:
This workflow assumes you’re familiar with Code Manager. It involves making changes to your control repo, such as adding or updating modules, editing manifests, or changing your Hiera data.

This workflow requires running deploy actions from the Code Manager command line tool and the orchestrator, so make sure you have access to a host with PE client tools installed.

Related information
Add code and set up Code Manager on page 161
Set up your control repo, create a Puppetfile, and configure Code Manager so you can start adding content to your Puppet Enterprise (PE) environments.

Enable cached catalogs for use with the orchestrator

Enabling cached catalogs on your agents ensures Puppet does not enforce any catalog changes on your agents until you run an orchestrator job to enforce changes.

When you use the orchestrator to enforce changes in a Puppet environment (for example, in your production environment), you want agents in that environment to maintain their cached catalogs until you run an orchestrator job that deploys configuration changes for those agents. In these environments, agents reinforce configuration from their cached catalogs during the normal run interval (30 minutes by default), and they apply new configuration only when you run Puppet with an orchestration job.

Important: Although enabling cached catalogs is optional (you can run Puppet on nodes with orchestrator workflows that don't require cached catalogs), our recommended workflow requires enabling cached catalogs so agents enforce cached catalogs by default and only compile new catalogs when instructed by orchestrator jobs.

1. Run Puppet on the new agents.

CAUTION: Run Puppet on new agents before assigning any application components to them or performing the next step.
2. In each agent’s puppet.conf file, add `use_cached_catalog=true` to the [agent] section. There are two ways to do this:

- From the command line on each agent machine, run:
  ```bash
  puppet config set use_cached_catalog true --section agent
  ```
- Add an ini_setting resource in the node default {} section of the environment’s site.pp file.

  **Important:** This adds the setting to all agents in the environment.

```ruby
if $facts['kernel'] == 'windows' {
  $config = 'C:/ProgramData/PuppetLabs/puppet/etc/puppet.conf'
} else {
  $config = $settings::config
}

ini_setting { 'use_cached_catalog':
  ensure => present,
  path => $config,
  section => 'agent',
  setting => 'use_cached_catalog',
  value => 'true',
}
```

3. Run Puppet on the agents again to enforce this configuration.

**Set up node groups for testing new features**

1. If they don’t already exist, create environment node groups for branch testing. For example, you could create Development environment and Test environment node groups.

2. Within each of these environment node groups, create a child node group to enable on-demand testing of changes deployed in Git feature branch Puppet environments.

You now have at three levels of environment node groups:

- The top-level parent environment node group
  - Node groups representing your actual environments
  - Node groups for feature testing

3. On the Rules tab for each feature testing child node group, add a rule with these values:

- **Fact:** agent_specified_environment
- **Operator:** ~
- **Value:** ^.+ 

This rule matches any nodes from the parent group that have the `agent_specified_environment` fact set. By matching nodes to this group, you give the nodes permission to override the server-specified environment and use their agent-specified environment instead.

**Related information**

Create environment node groups on page 461
Create custom environment node groups so that you can target deployment of Puppet code.

**Create a feature branch**

After setting up a node group for feature testing, create a feature branch in your control repository. A feature branch allows you to develop and test code before merging it with the main branch.

1. Create a feature branch in your control repository, and name the branch clearly as a feature branch (for example, `my_feature_branch` or `feature_<TICKET_NUMBER>`).

2. Make changes to the code on the feature branch, and commit and push the changes to the feature branch.
Deploy code to the primary server and test it

After making changes to the code on your feature branch, use Code Manager to push those changes to the primary server.

1. To deploy the code from the feature branch to the primary server, run this Code Manager command:

```
puppet code deploy --wait <FEATURE_BRANCH>
```

After this code deployment, wait while Puppet Server loads the new code. The primary server now has code from the main/production branch and the feature branch.
2. To test your changes, use this orchestrator command to run Puppet on a few agent-specified development nodes in the feature branch environment:

```
puppet job run --nodes <DEV-NODE1>,<DEV_NODE2> --environment <FEATURE_BRANCH>
```

**Tip:** You can also use the console to create a job targeting a list of nodes in the feature branch environment.

At this point, the production environment maintains the original code deployed to all production nodes, and the feature branch environment has deployed your new code from the feature branch to the nodes you specified.

3. Review the node run reports in the console by opening the links in the orchestrator command output or using the Job ID linked on the Job list page. Verify the code changes had the intended effect.

**Related information**

- [Run Puppet on a node list](#) on page 621
Create a node list target for a job when you need to run Puppet on a specific set of nodes that isn't easily defined by a PQL query.

**Merge and promote your code**

If everything works as expected on the development nodes, you're ready to promote your changes into production.

1. Merge your feature branch into the production branch in your control repo.

2. To deploy your updated production branch to the primary server, run this Code Manager command:

   ```bash
   puppet code deploy --wait production
   ```

**Preview the job**

Before running Puppet across the production environment, use the `puppet job plan` command to preview the job.

1. To ensure the job captures all nodes in the production environment, as well as the agent-specified development nodes that just ran with the feature branch environment, use this query as the job target:

   ```bash
   puppet job plan --query 'inventory {environment in ["production", "<FEATURE_BRANCH>"]}'
   ```

2. Review the outcome to ensure you want to make these changes.
Run the job on the production environment

If you're satisfied with the job preview, you're ready to enforce changes on the production environment.

1. To deploy the new code to the primary server and all nodes in the production environment, run the orchestrator job:

   ```bash
   puppet job run --query 'inventory {environment in ["production", 
   
   "<FEATURE_BRANCH>"]}'
   ```

   **Tip:** You can also use the console to create a job targeted at this PQL query.

2. Check the node run reports in the console to confirm that the changes were applied as intended.

Repeat this process as you develop and promote your code.

**Related information**

Run Puppet on a PQL query on page 622
For some jobs, you might want to target nodes that meet specific conditions. For such jobs, create a PQL query.

**Configuring Puppet orchestrator**

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

- Set PE RBAC permissions and token authentication for Puppet orchestrator
- Enable cached catalogs for use with the orchestrator (optional)
- Review the orchestrator configuration files and adjust them as needed

All of these instructions assume that PE client tools are installed.

**Related information**

*Installing client tools* on page 250

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

**Orchestration services settings**

**Global logging and SSL settings**

`/etc/puppetlabs/orchestration-services/conf.d/global.conf` contains settings shared across the Puppet Enterprise (PE) orchestration services.

The file `global.certs` typically requires no changes and contains the following settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl-cert</td>
<td>Certificate file path for the orchestrator host.</td>
<td><code>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.cert.pem</code></td>
</tr>
<tr>
<td>ssl-key</td>
<td>Private key path for the orchestrator host.</td>
<td><code>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.private_key.pem</code></td>
</tr>
<tr>
<td>ssl-ca-cert</td>
<td>CA file path</td>
<td><code>/etc/puppetlabs/puppet/ssl/ca.pem</code></td>
</tr>
</tbody>
</table>

The file `global.logging-config` is a path to `logback.xml` file that configures logging for most of the orchestration services. See [http://logback.qos.ch/manual/configuration.html](http://logback.qos.ch/manual/configuration.html) for documentation on the structure of the `logback.xml` file. It configures the log location, rotation, and formatting for the following:

- orchestration-services (appender section F1)
- orchestration-services status (STATUS)
- pcp-broker (PCP)
- pcp-broker access (PCP_ACCESS)
- aggregate-node-count (AGG_NODE_COUNT)

**Allow list of trapperkeeper services to start**

`/etc/puppetlabs/orchestration-services/bootstrap.cfg` is the list of trapperkeeper services from the orchestrator and pcp-broker projects that are loaded when the `pe-orchestration-services` system service starts.
• To disable a service in this list, remove it or comment it with a # character and restart `pe-orchestration-services`.
• To enable an NREPL service for debugging, add `puppetlabs.trapperkeeper.services.nrepl.nrepl-service/nrepl-service` to this list and restart `pe-orchestration-services`.

The **pcp-broker and orchestrator HTTP services**

`/etc/puppetlabs/orchestration-services/conf.d/webserver.conf` describes how and where to run pcp-broker and orchestrator web services, which accept HTTP API requests from the rest of the PE installation and from external nodes and users.

The file `webserver.orchestrator` configures the orchestrator web service. Defaults are as follows:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-log-config</td>
<td>A logback XML file configuring logging for orchestrator access messages.</td>
<td><code>/etc/puppetlabs/orchestration-services/request-logging.xml</code></td>
</tr>
<tr>
<td>client-auth</td>
<td>Determines the mode that the server uses to validate the client's certificate for incoming SSL connections.</td>
<td><code>want</code> or <code>need</code></td>
</tr>
<tr>
<td>default-server</td>
<td>Allows multi-server configurations to run operations without specifying a server-id. Without a server-id, operations will run on the selected default. Optional.</td>
<td><code>true</code></td>
</tr>
<tr>
<td>ssl-ca-cert</td>
<td>Sets the path to the CA certificate PEM file used for client authentication.</td>
<td><code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code></td>
</tr>
<tr>
<td>ssl-cert</td>
<td>Sets the path to the server certificate PEM file used by the web service for HTTPS.</td>
<td><code>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.cert.pem</code></td>
</tr>
<tr>
<td>ssl-crl-path</td>
<td>Describes a path to a Certificate Revocation List file. Optional.</td>
<td><code>/etc/puppetlabs/puppet/ssl/crl.pem</code></td>
</tr>
<tr>
<td>ssl-host</td>
<td>Sets the host name to listen on for encrypted HTTPS traffic.</td>
<td><code>0.0.0.0.</code></td>
</tr>
<tr>
<td>ssl-key</td>
<td>Sets the path to the private key PEM file that corresponds with the ssl-cert</td>
<td><code>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.private_key.pem</code></td>
</tr>
<tr>
<td>ssl-port</td>
<td>Sets the port to use for encrypted HTTPS traffic.</td>
<td><code>8143</code></td>
</tr>
</tbody>
</table>

The file `webserver.pcp-broker` configures the pcp-broker web service. Defaults are as follows:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>client-auth</td>
<td>Determines the mode that the server uses to validate the client's certificate for incoming SSL connections.</td>
<td><code>want</code> or <code>need</code></td>
</tr>
<tr>
<td>Setting</td>
<td>Definition</td>
<td>Default</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>ssl-ca-cert</td>
<td>Sets the path to the CA certificate PEM file used for client authentication.</td>
<td>/etc/puppetlabs/puppet/ssl/certs/ca.pem</td>
</tr>
<tr>
<td>ssl-cert</td>
<td>Sets the path to the server certificate PEM file used by the web service for HTTPS.</td>
<td>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.cert.pem</td>
</tr>
<tr>
<td>ssl-crl-path</td>
<td>Describes a path to a Certificate Revocation List file. Optional.</td>
<td>/etc/puppetlabs/puppet/ssl/crl.pem</td>
</tr>
<tr>
<td>ssl-host</td>
<td>Sets the host name to listen on for encrypted HTTPS traffic.</td>
<td>0.0.0.0.</td>
</tr>
<tr>
<td>ssl-key</td>
<td>Sets the path to the private key PEM file that corresponds with the ssl-cert.</td>
<td>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.private_key.pem</td>
</tr>
<tr>
<td>ssl-port</td>
<td>Sets the port to use for encrypted HTTPS traffic.</td>
<td>8142</td>
</tr>
</tbody>
</table>

/etc/puppetlabs/orchestration-services/conf.d/web-routes.conf describes how to route HTTP requests made to the API web servers, designating routes for interactions with other services. These should not be modified. See the configuration options at the trapperkeeper-webserver-jetty project's docs.

### Analytics trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/analytics.conf contains the internal setting for the analytics trapperkeeper service.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Definition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>analytics.url</td>
<td>Specifies the API root.</td>
<td>&lt;puppetserver-host-url&gt;:8140/analytics/v1</td>
</tr>
</tbody>
</table>

### Authorization trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/auth.conf contains internal settings for the authorization trapperkeeper service. See configuration options in the trapperkeeper-authorization project's docs.

### JMX metrics trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/metrics.conf contains internal settings for the JMX metrics service built into orchestration-services. See the service configuration options in the trapperkeeper-metrics project's docs.

### Orchestrator trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/orchestrator.conf contains internal settings for the orchestrator project's trapperkeeper service.

### PCP broker trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/pcp-broker.conf contains internal settings for the pcp-broker project's trapperkeeper service. See the service configuration options in the pcp-broker project's docs.
Related information

Configure the orchestrator and pe-orchestration-services on page 303

These are some optional parameters you can use to configure the behavior of the orchestrator and the pe-orchestration-services service.

Orchestrator configuration files

The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate workstation, there are two types of configuration files: a global configuration file and a user-specified configuration file.

Orchestrator global configuration file

If you’re running the orchestrator from a PE-managed machine, on either the primary server or an agent node, PE manages the global configuration file.

This file is installed on both managed and non-managed workstations at:

- *nix systems --- /etc/puppetlabs/client-tools/orchestrator.conf
- Windows --- C:/ProgramData/PuppetLabs/client-tools/orchestrator.conf

The class that manages the global configuration file is puppet_enterprise::profile::controller. The following parameters and values are available for this class:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage_orchestrator</td>
<td>true or false (default is true)</td>
</tr>
<tr>
<td>orchestrator_url</td>
<td>url and port (default is primary server url and port 8143)</td>
</tr>
</tbody>
</table>

The only value PE sets in the global configuration file is the orchestrator_url (which sets the orchestrator's service-url in /etc/puppetlabs/client-tools/orchestrator.conf).

Important: If you're using a managed workstation, do not edit or change the global configuration file. If you're using an unmanaged workstation, you can edit this file as needed.

Orchestrator user-specified configuration file

You can manually create a user-specified configuration file and populate it with orchestrator configuration file settings. PE does not manage this file.

This file needs to be located at ~/.puppetlabs/client-tools/orchestrator.conf for both *nix and Windows.

If present, the user specified configuration always takes precedence over the global configuration file. For example, if both files have contradictory settings for the environment, the user specified settings prevail.

Orchestrator configuration file settings

The orchestrator configuration file is formatted in JSON. For example:

```json
{
   "options" : {
      "service-url": "https://<PRIMARY SERVER HOSTNAME>:8143",
      "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
      "token-file": "~/.puppetlabs/token",
      "color": true
   }
}
```

The orchestrator configuration files (the user-specified or global files) can take the following settings:
<table>
<thead>
<tr>
<th>Setting</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service-url</td>
<td>The URL that points to the primary server and the port used to communicate with the orchestration service. (You can set this with the orchestrator_url parameter in the puppet_enterprise::profile::controller class.) Default value: https://&lt;PRIMARY_SERVER_HOSTNAME&gt;:8143</td>
</tr>
<tr>
<td>environment</td>
<td>The environment used when you issue commands with Puppet orchestrator.</td>
</tr>
<tr>
<td>cacert</td>
<td>The path for the Puppet Enterprise CA cert.</td>
</tr>
<tr>
<td>*nix: /etc/puppetlabs/puppet/ssl/certs/ca.pem</td>
<td>*nix: /etc/puppetlabs/puppet/ssl/certs/ca.pem</td>
</tr>
<tr>
<td>Windows: C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem</td>
<td>Windows: C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem</td>
</tr>
<tr>
<td>token-file</td>
<td>The location for the authentication token. Default value: ~/.puppetlabs/token</td>
</tr>
<tr>
<td>color</td>
<td>Determines whether the orchestrator output uses color. Set to true or false.</td>
</tr>
<tr>
<td>noop</td>
<td>Determines whether the orchestrator runs the Puppet agent in no-op mode. Set to true or false.</td>
</tr>
</tbody>
</table>

**Setting PE RBAC permissions and token authentication for orchestrator**

Before you run any orchestrator jobs, you need to set the appropriate permissions in PE role-based access control (RBAC) and establish token-based authentication.

Most orchestrator users require the following permissions to run orchestrator jobs or tasks:

<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet agent</td>
<td>Run Puppet on agent nodes</td>
<td>The ability to run Puppet on nodes using the console or orchestrator. Instance must always be &quot;*&quot;.</td>
</tr>
<tr>
<td>Job orchestrator</td>
<td>Start, stop and view jobs</td>
<td>The ability to start and stop jobs and tasks, view jobs and job progress, and view an inventory of nodes that are connected to the PCP broker.</td>
</tr>
<tr>
<td>Tasks</td>
<td>Run tasks</td>
<td>The ability to run specific tasks on all nodes, a selected node group, or nodes that match a PQL query.</td>
</tr>
<tr>
<td>Nodes</td>
<td>View node data from PuppetDB</td>
<td>The ability to view node data imported from PuppetDB. Object must always be &quot;*&quot;.</td>
</tr>
</tbody>
</table>
**Note:** If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

**Assign task permissions to a user role**

1. In the console, on the Access control page, click the User roles tab.
2. From the list of user roles, click the one you want to have task permissions.
3. On the Permissions tab, in the Type box, select Tasks.
4. For Permission, select Run tasks, and then select a task from the Object list. For example, facter_task.
5. Click Add permission, and then commit the change.

**Using token authentication**

Before running an orchestrator job, you must generate an RBAC access token to authenticate to the orchestration service. If you attempt to run a job without a token, PE prompts you to supply credentials.

For information about generating a token with the CLI, see the documentation on token-based authentication.

**Related information**

Token-based authentication on page 347
Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Create a new user role on page 339
RBAC has four predefined roles: Administrators, Code Deployers, Operators, and Viewers. You can also define your own custom user roles.

Assign permissions to a user role on page 339
You can mix and match permissions to create custom user roles that provide users with precise levels of access to PE actions.

**PE Bolt server configuration**

The PE Bolt server provides an API for running tasks over SSH and WinRM using Bolt, which is the technology underlying PE tasks. You do not need to have Bolt installed to configure the Bolt server or run tasks in PE. The API server for tasks is available as pe-bolt-server.

The server is a Puma application that runs as a standalone service.

The server is configured in /etc/puppetlabs/bolt-server/conf.d/bolt-server.conf, managed by the puppet enterprise::profile::bolt_server class, which includes these parameters:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bolt_server_loglevel</td>
<td>string</td>
<td>Bolt log level. Acceptable values are debug, info, notice, warn, and error.</td>
<td>notice</td>
</tr>
<tr>
<td>concurrency</td>
<td>integer</td>
<td>Maximum number of server threads.</td>
<td>100</td>
</tr>
<tr>
<td>master_host</td>
<td>string</td>
<td>URI of the primary server where Bolt can download tasks.</td>
<td>$puppet enterprise::puppet_master_host</td>
</tr>
<tr>
<td>master_port</td>
<td>integer</td>
<td>Port the Bolt server can access the primary server on.</td>
<td>$puppet enterprise::puppet_master_port</td>
</tr>
</tbody>
</table>
PE ACE server configuration

The PE ACE server is a service that allows for tasks and catalogs to run against remote targets.

The server is a Puma application that runs as a standalone service.

The server is configured in /etc/puppetlabs/ace-server/conf.d/ace-server.conf and managed in the puppet_enterprise::profile::ace_server class, which includes these parameters:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_loglevel</td>
<td>string</td>
<td>Bolt log level. Acceptable values are debug, info, notice, warn, and error.</td>
<td>notice</td>
</tr>
<tr>
<td>concurrency</td>
<td>integer</td>
<td>Maximum number of server threads.</td>
<td>$puppet_enterprise::ace_server_concurrency</td>
</tr>
<tr>
<td>master_host</td>
<td>string</td>
<td>URI that ACE can access the primary server on.</td>
<td>pe_repo::compile_master_pool_address::default: $puppet_enterprise::puppet_master_host</td>
</tr>
<tr>
<td>master_port</td>
<td>integer</td>
<td>Port that ACE can access the primary server on.</td>
<td>$puppet_enterprise::puppet_master_port</td>
</tr>
<tr>
<td>hostcrl</td>
<td>string</td>
<td>The host CRL path</td>
<td>$puppet_enterprise::params::hostcrl</td>
</tr>
<tr>
<td>ssl_cipher_suites</td>
<td>array[string]</td>
<td>TLS cipher suites in order of preference.</td>
<td>$puppet_enterprise::params::secure_ciphers</td>
</tr>
<tr>
<td>ssl_listen_port</td>
<td>integer</td>
<td>Port that ACE runs on.</td>
<td>44633 ($puppet_enterprise::ace_server_port)</td>
</tr>
<tr>
<td>allowlist</td>
<td>array[string]</td>
<td>List of hosts that can connect to pe-ace-server.</td>
<td>[$certname]</td>
</tr>
</tbody>
</table>

Running Puppet on demand

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

- **Running Puppet on demand from the console** on page 621

When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

- **Running Puppet on demand from the CLI** on page 627

Use the puppet job run command to enforce change on your agent nodes with on-demand Puppet runs.

- **Running Puppet on demand with the API** on page 630
Run the orchestrator across all nodes in an environment.

Running Puppet on demand from the console

When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

There are two ways to specify the job target (the nodes you want to run jobs on):

- A static node list
- A Puppet Query Language (PQL) query
- A node group

You can't combine these methods, and if you switch from one to the other with the Inventory drop-down list, the target list clears and starts over. In other words, if you create a target list by using a node list, switching to a PQL query clears the node list, and vice versa. You can do a one-time conversion of a PQL query to a static node list if you want to add or remove nodes from the query results.

Before you start, be sure you have the correct permissions for running jobs. To run jobs on PQL queries, you need the "View node data from PuppetDB” permission.

Run Puppet on a node list

Create a node list target for a job when you need to run Puppet on a specific set of nodes that isn't easily defined by a PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run jobs and PQL queries.

Tip: You can add network devices to a node list when you have installed modules for device transports in your production environment. You can find such modules in Puppet Forge.

1. In the console, on the Jobs page, click Run Puppet.
2. Optional: In the Job description field, provide a description. The text you enter here appears on the job list and job details pages.
3. Optional: Under Schedule, select Later and choose a start date, time, time zone, and frequency for the job to run.
4. Select an environment:
   - Run nodes in their own assigned environment: Nodes run in the environment specified by the Node Manager or their puppet.conf file.
   - Select an environment for nodes to run in: Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
   - No-op: Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
   - Debug: Print all debugging messages.
   - Trace: Print stack traces on some errors.
   - Eval-trace: Display how long it took for each step to run.
   - Override noop = true configuration: If any nodes in the job have noop = true set in their puppet.conf files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the --no-noop flag available on the orchestrator CLI.
6. In the Inventory list, select Node list.
7. To create a node list, in the search field, start typing in the names of nodes to search for, and click **Search**.

   **Note:** The search does not handle regular expressions.

8. Select the nodes you want to add to the job. You can select nodes from multiple searches to create the node list target.

9. Click **Run job**.

   View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only on the nodes that failed during the initial run.

**Run Puppet on a PQL query**

For some jobs, you might want to target nodes that meet specific conditions. For such jobs, create a PQL query.

**Before you begin**

- Make sure you have access to the nodes you want to target.
- Make sure you have permissions to run jobs and PQL queries.

1. In the console, on the **Jobs** page, click **Run Puppet**.

2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

3. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

4. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes run in the environment specified by the Node Manager or their puppet.conf file.
   - **Select an environment for nodes to run in**: Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.

5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
   - **No-op**: Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
   - **Debug**: Print all debugging messages.
   - **Trace**: Print stack traces on some errors.
   - **Eval-trace**: Display how long it took for each step to run.
   - **Override noop = true configuration**: If any nodes in the job have `noop = true` set in their puppet.conf files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the **--no-noop** flag available on the orchestrator CLI.

6. From the list of target types, select **PQL query**.
7. Specify a target by doing one of the following:
   - Enter a query that selects the target you want. See the Puppet Query Language (PQL) reference for more information.
   - Click **Common queries**. Select one of the queries and replace the defaults in the braces ({} ) with values that specify the target you want.

   **Note:** These queries include [certname] as [<projection>] to restrict the output.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes</td>
<td>nodes[certname] ( )</td>
</tr>
<tr>
<td>Nodes with a specific resource (example: httpd)</td>
<td>resources[certname] ( type = &quot;Service&quot; and title = &quot;httpd&quot; )</td>
</tr>
<tr>
<td>Nodes with a specific fact and value (example: OS name is CentOS)</td>
<td>inventory[certname] ( facts.os.name = &quot;&lt;OS&gt;&quot; )</td>
</tr>
<tr>
<td>Nodes with a specific report status (example: last run failed)</td>
<td>reports[certname] ( latest_report_status = &quot;failed&quot; )</td>
</tr>
<tr>
<td>Nodes with a specific class (example: Apache)</td>
<td>resources[certname] ( type = &quot;Class&quot; and title = &quot;Apache&quot; )</td>
</tr>
<tr>
<td>Nodes assigned to a specific environment (example: production)</td>
<td>nodes[certname] ( catalog_environment = &quot;production&quot; )</td>
</tr>
<tr>
<td>Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)</td>
<td>resources[certname] (type = &quot;Package&quot; and title=&quot;openssl&quot; and parameters.ensure = &quot;1.0.1e-51.el7_2.7&quot; )</td>
</tr>
<tr>
<td>Nodes with a specific resource and operating system (example: httpd and CentOS)</td>
<td>inventory[certname] ( facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } )</td>
</tr>
</tbody>
</table>

8. Click **Submit query** and click **Refresh** to update the node results.
9. If you change or edit the query after it runs, click **Submit query** again.

10. Optional: To convert the PQL query target results to a node list, for use as a node list target, click **Convert query to static node list**.

   **Note:** If you select this option, the job target becomes a node list. You can add or remove nodes from the node list before running the job, but you cannot edit the query.

11. Click **Run job**.
    View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only on the nodes that failed during the initial run.

   **Important:** When you run this job, the PQL query runs again, and the job might run on a different set of nodes than what is currently displayed. If you want the job to run only on the list as displayed, convert the query to a static node list before you run the job.
Add custom PQL queries to the console
Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary server, copy the `custom_pql_queries.json.example` file and remove the `.example` suffix.
   ```bash
   sudo cp /etc/puppetlabs/console-services/custom_pql_queries.json.example /etc/puppetlabs/console-services/custom_pql_queries.json
   ```

2. Edit the file contents to include your own PQL queries or remove any existing queries.
3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop down options when running jobs.

Run Puppet on a node group
Create a node target for a job when you need to run Puppet on a specific set of nodes in a pre-defined group.

Before you begin
Make sure you have access to the nodes you want to target.

Note: If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

1. In the console, on the Jobs page, click Run Puppet.
2. Optional: In the Job description field, provide a description. The text you enter here appears on the job list and job details pages.
3. Optional: Under Schedule, select Later and choose a start date, time, time zone, and frequency for the job to run.
4. Select an environment:
   - Run nodes in their own assigned environment: Nodes run in the environment specified by the Node Manager or their puppet.conf file.
   - Select an environment for nodes to run in: Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
   - No-op: Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
   - Debug: Print all debugging messages.
   - Trace: Print stack traces on some errors.
   - Eval-trace: Display how long it took for each step to run.
   - Override noop = true configuration: If any nodes in the job have `noop = true` set in their puppet.conf files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.
6. From the list of target types, select Node group.
7. In the Choose a node group box, type or select a node group, and click Select.
8. Click Run job.

View the job status and a list of previous Puppet jobs on the Jobs page. To rerun the job, click Run again and choose to rerun it on all nodes or only on the nodes that failed during the initial run.
Run jobs throughout the console

You don't need to be in the Jobs section of the console to run a Puppet job on your nodes. You might encounter situations where you want to run jobs on lists of nodes derived from different pages in the console.

You can create jobs from the following pages:

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>This page shows a list of all your managed nodes, and gathers essential information about your infrastructure at a glance.</td>
</tr>
<tr>
<td>Events</td>
<td>Events let you view a summary of activity in your infrastructure, analyze the details of important changes, and investigate common causes behind related events. For instance, let's say you notice run failures because some nodes have out-of-date code. After you update the code, you can create a job target from the list of failed nodes to be sure you're directing the right fix to the right nodes. You can create new jobs from the Nodes with events category.</td>
</tr>
<tr>
<td>Classification node groups</td>
<td>Node groups are used to automate classification of nodes with similar functions in your infrastructure. If you make a classification change to a node group, you can quickly create a job to run Puppet on all the nodes in that group, pushing the change to all nodes at one time.</td>
</tr>
</tbody>
</table>

Make sure you have permissions to run jobs and PQL queries.

1. In the console, in the Run section, click Puppet.
   At this point, the list of nodes is converted to a new Puppet run list target.

2. Optional: In the Job description field, provide a description. The text you enter here appears on the job list and job details pages.

3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes run in the environment specified by the Node Manager or their puppet.conf file.
   - **Select an environment for nodes to run in**: Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.

4. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
   - **No-op**: Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
   - **Debug**: Print all debugging messages.
   - **Trace**: Print stack traces on some errors.
   - **Eval-trace**: Display how long it took for each step to run.
   - **Override noop = true configuration**: If any nodes in the job have noop = true set in their puppet.conf files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the --no-noop flag available on the orchestrator CLI.

5. **Important**: Do not change the Inventory from Node list to PQL query. This clears the node list target.

6. Click Run job.
   View the job status and a list of previous Puppet jobs on the Jobs page. To rerun the job, click Run again and choose to rerun the it on all nodes or only on the nodes that failed during the initial run.
Schedule a Puppet run
Schedule a job to deploy configuration changes at a particular date and time.

Before you begin
Make sure you have access to the nodes you want to target.

**Note:** If a reboot occurs or if you need to restore a backup, scheduled Puppet jobs are rescheduled based on the last execution time. If the scheduled Puppet job running in the orchestrator is what caused the reboot, the job appears as failed.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
   - **Select an environment for nodes to run in**: Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
4. Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
5. Optional: To repeat the job on a regular schedule, change the run frequency from **Once** to **Hourly**, **Daily**, or **Weekly**.
   
   **Note:** If a recurring job runs longer than the selected frequency, the following job waits to start until the next available time interval.

6. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
   - **No-op**: Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
   - **Debug**: Print all debugging messages.
   - **Trace**: Print stack traces on some errors.
   - **Eval-trace**: Display how long it took for each step to run.
   - **Override noop = true configuration**: If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.

7. From the list of target types, select the category of nodes you want to target.
   - **Node list**: Add individual nodes by name.
   - **PQL Query**: Use the Puppet query language to retrieve a list of nodes.
   - **Node group**: Select an existing node group.

8. Click **Schedule job**.
   
   Your job appears on the **Scheduled Puppet run** tab of the **Jobs** page.

Stop a job in progress
You can stop a job if, for example, you realize you need to reconfigure a class or push a configuration change that the job needs.

When you stop a Puppet job, jobs that are underway finish, and jobs that have not started are canceled.

To stop a job:
   - In the console, go to the **Jobs** page and select the **Puppet run** tab. From the list of jobs, find the one you want and click **Stop**.
   - On the command line, press **CTRL + C**.
Delete a scheduled job
Delete a job that is scheduled to run at a later time.

Tip: You must have the appropriate role-based permissions to delete another user's scheduled job.

1. In the console, open the Jobs page.
2. Click the Scheduled Puppet Run tab.
3. From the list of jobs, find the one you want to delete and click Remove.

Running Puppet on demand from the CLI
Use the puppet job run command to enforce change on your agent nodes with on-demand Puppet runs.

Use the puppet job run command to enforce change across nodes. For example, when you add a new class parameter to a set of nodes or deploy code to a new Puppet environment, use this command to run Puppet across all the nodes in that environment.

You can run jobs on three types of targets, but these targets cannot be combined:
- A list of nodes or a single node
- A node group
- A PQL nodes query

When you execute a puppet job run command, the orchestrator creates a new Job ID, shows you all nodes included in the job, and proceeds to run Puppet on all nodes in the appropriate order. Puppet compiles a new catalog for all nodes included in the job.

See puppet job run command options on page 629 for additional run options.

The first time you run a command, you need to authenticate. See the orchestrator installation instructions for information about setting RBAC permissions and token authorization.

If you're running this command from a managed or non-managed Windows workstation, you must specify the full path to the command. For example, c:\Program Files\Puppet Labs\Client\bin\puppet-task run.

Run Puppet on a list of nodes or a single node
Use a node list target for an orchestrator job when you need to run a job on a specific set of nodes that don't easily resolve to a PQL query. Use a single node or a comma-separated list of nodes.

Before you begin
Make sure you have permissions to run jobs.
Make sure you have access to the nodes you want to target.

Log into your primary server or client tools workstation and run one of the following commands:
- To run a job on a single node:
  puppet job run --nodes <NODE NAME> <OPTIONS>
- To run a job on a list of nodes, use a comma-separated list of node names:
  puppet job run --nodes <NODE NAME>,<NODE NAME>,<NODE NAME>,<NODE NAME> <OPTIONS>

Note: Do not add spaces in the list of nodes.
To run a job on a node list from a text file:

```
puppet job run --nodes @/path/to/file.txt
```

**Note:** If passing a list of nodes in the text file, put each node on a separate line.

To view the status of the Puppet job, run `puppet job show <job ID>`. To view a list of the previous 50 running and complete Puppet jobs, run `puppet job show`.

### Run Puppet on a PQL query

Use a PQL nodes query as a target when you want to target nodes that meet specific conditions. In this case, the orchestrator runs on a list of nodes returned from a PQL nodes query.

#### Before you begin

- Make sure you have access to the nodes you want to target.
- Make sure you have permissions to run jobs and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- To specify the query on the command line:
  ```
puppet job run --query '<QUERY>' <OPTIONS>
  ```
- To pass the query in a text file:
  ```
puppet job run --query @/path/to/file.txt
  ```

The following table shows some example targets and the associated PQL queries you could run with the orchestrator.

Be sure to wrap the entire query in single quotes and use double quotes inside the query.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single node by certname</td>
<td>--query 'nodes { certname = &quot;mynode&quot; }'</td>
</tr>
<tr>
<td>All nodes with &quot;web&quot; in certname</td>
<td>--query 'nodes { certname ~ &quot;web&quot; }'</td>
</tr>
<tr>
<td>All CentOS nodes</td>
<td>--query 'inventory { facts.os.name = &quot;CentOS&quot; }'</td>
</tr>
<tr>
<td>All CentOS nodes with httpd managed</td>
<td>--query 'inventory { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }'</td>
</tr>
<tr>
<td>All nodes with failed reports</td>
<td>--query 'reports { latest_report? = true and status = &quot;failed&quot; }'</td>
</tr>
<tr>
<td>All nodes matching the environment for the last received catalog</td>
<td>--query 'nodes { catalog_environment = &quot;production&quot; }'</td>
</tr>
</tbody>
</table>

To view the status of the Puppet job, run `puppet job show <job ID>`. To view a list of the previous 50 running and complete Puppet jobs, run `puppet job show`.

### Run Puppet on a node group

Similar to running Puppet on a list of nodes, you can run it on a node group.

#### Before you begin

- Make sure you have permissions to run jobs.
- Make sure you have access to the nodes you want to target.

1. Log into your primary server or client tools workstation.
2. Run the command: `puppet job run --node-group <node-group-id>`

   Tip: Use the `/v1/groups` endpoint to retrieve a list node groups and their IDs.

Related information
GET `/v1/groups` on page 530
Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

`puppet job run` command options
The following are common options you can use with the `run` action. For a complete list of global options run `puppet job --help`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--noop</code></td>
<td>Flag, default false</td>
<td>Run a job on all nodes to simulate changes from a new catalog without actually enforcing a new catalog. Cannot be used in conjunction with <code>--no-noop</code> flag.</td>
</tr>
<tr>
<td><code>--no-noop</code></td>
<td>Flag</td>
<td>All nodes run in enforcement mode, and a new catalog is enforced on all nodes. This flag overrides the agent <code>noop = true</code> in <code>puppet.conf</code>. Cannot be used in conjunction with <code>--noop</code> flag.</td>
</tr>
<tr>
<td><code>--environment</code>, <code>-e</code></td>
<td>Environment name</td>
<td>Overrides the environment specified in the orchestrator configuration file. The orchestrator uses this option to tell nodes which environment to run in. If any nodes can’t run in the environment, those node runs fail. A node runs in an environment as long as it is classified into that environment in the PE node classifier.</td>
</tr>
<tr>
<td><code>--no-enforce-environment</code></td>
<td>Flag, default false</td>
<td>Ignores the environment set by the <code>--environment</code> flag for agent runs. When you use this flag, agents run in the environment specified by the PE Node Manager or their <code>puppet.conf</code> files.</td>
</tr>
<tr>
<td><code>--description</code></td>
<td>Flag, defaults to empty</td>
<td>Provide a description for the job, to be shown on the job list and job details pages, and returned with the <code>puppet job show</code> command.</td>
</tr>
<tr>
<td><code>--concurrency</code></td>
<td>Integer</td>
<td>Limits how many nodes can run concurrently. Default is unlimited. You can tune concurrent compile requests in the console.</td>
</tr>
</tbody>
</table>

Post-run node status
After Puppet runs, the orchestrator returns a list of nodes and their run statuses.

Node runs can be in progress, completed, skipped, or failed.
• For a completed node run, the orchestrator prints the configuration version, the transaction ID, a summary of resource events, and a link to the full node run report in the console.
• For an in progress node run, the orchestrator prints how many seconds ago the run started.
• For a failed node run, the orchestrator prints an error message indicating why the run failed. In this case, any additional runs are skipped.

When a run fails, the orchestrator also prints any applications that were affected by the failure, as well as any applications that were affected by skipped node runs.

You can view the status of all running, completed, and failed jobs with the puppet job show command, or you can view them from the Job details page in the console.

Additionally, on the Jobs page in the console, you can review a list of jobs or to view the details of jobs that have previously run or are in progress.

Stop a job in progress
You can stop a job if, for example, you realize you need to reconfigure a class or push a configuration change that the job needs.

When you stop a Puppet job, jobs that are underway finish, and jobs that have not started are canceled.

To stop a job:
• In the console, go to the Jobs page and select the Puppet run tab. From the list of jobs, find the one you want and click Stop.
• On the command line, press CTRL + C.

POST /command/deploy
Run the orchestrator across all nodes in an environment.

Request format
The request body must be a JSON object using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>The environment to deploy. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>scope</td>
<td>Object, required unless target is specified. The PuppetDB query, a list of nodes, or a classifier node group id to deploy.</td>
</tr>
<tr>
<td>description</td>
<td>String, a description of the job.</td>
</tr>
<tr>
<td>noop</td>
<td>Boolean, whether to run the agent in no-op mode. The default is false.</td>
</tr>
<tr>
<td>no_noop</td>
<td>Boolean, whether to run the agent in enforcement mode. Defaults to false. This flag overrides noop = true if set in the agent's puppet.conf and cannot be set to true at the same time as the noop flag.</td>
</tr>
<tr>
<td>concurrency</td>
<td>The maximum number of nodes to run at one time. The default is a range between 1 and the value of the global_concurrent_compiles parameter. For information about the global_concurrent_compiles parameter, refer to Configuring orchestration on page 302.</td>
</tr>
</tbody>
</table>
### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>enforce_environment</td>
<td>Boolean, whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be false if environment is an empty string.</td>
</tr>
<tr>
<td>debug</td>
<td>Boolean, whether to use the --debug flag on Puppet agent runs.</td>
</tr>
<tr>
<td>trace</td>
<td>Boolean, whether to use the --trace flag on Puppet agent runs.</td>
</tr>
<tr>
<td>evaltrace</td>
<td>Boolean, whether to use the --evaltrace flag on Puppet agent runs.</td>
</tr>
<tr>
<td>filetimeout</td>
<td>Integer, sets the --filetimeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_connect_timeout</td>
<td>Integer, sets the --http_connect_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_keepalive_timeout</td>
<td>Integer, sets the --http_keepalive_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_read_timeout</td>
<td>Integer, sets the --http_read_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>ordering</td>
<td>String, sets the --ordering flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>skip_tags</td>
<td>String, sets the --skip_tags flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>tags</td>
<td>String, sets the --tags flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>use_cached_catalog</td>
<td>Boolean, whether to use the --use_cached_catalog flag on Puppet agent runs.</td>
</tr>
<tr>
<td>usecacheonfailure</td>
<td>Boolean, whether to use the --usecacheonfailure flag on Puppet agent runs.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

For example, to deploy the node1.example.com environment in no-op mode, the following request is valid:

```json
{
    "environment": "",
    "enforce_environment": false,
    "noop": true,
    "scope": {
        "nodes": ["node1.example.com"]
    },
    "userdata": {
        "servicenow_ticket": "INC0011211"
    }
}
```

### Scope

Scope is a JSON object containing exactly one of these keys:
### Key | Definition
--- | ---
application | The name of an application or application instance to deploy. If an application type is specified, all instances of that application are deployed.
nodes | A list of node names to target.
query | A PuppetDB or PQL query to use to discover nodes. The target is built from `certname` values collected at the top level of the query.
node_group | A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group.

To deploy an application instance in the `production` environment:

```json
{
    "environment" : "production",
    "scope" : {
        "application" : "Wordpress_app[demo]"
    }
}
```

To deploy a list of nodes:

```json
{
    "environment" : "production",
    "scope" : {
        "nodes" : ["node1.example.com", "node2.example.com"]
    }
}
```

To deploy a list of nodes with the `certname` value matching a regex:

```json
{
    "environment" : "production",
    "scope" : {
        "query" : ["from", "nodes", ["~", "certname", ".+"]]
    }
}
```

To deploy to the nodes defined by the "All Nodes" node group:

```json
{
    "environment" : "production",
    "scope" : {
        "node_group" : "00000000-0000-4000-8000-000000000000"
    }
}
```

### Response format

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response is a JSON object containing a link to retrieve information about the status of the job and uses any one of these keys:
For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
    "name" : "1234"
  }
}
```

**Error responses**

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the environment does not exist, the server returns a 404 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-environment</td>
<td>If the environment requested contains no applications or no nodes, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-target</td>
<td>If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/dependency-cycle</td>
<td>If the application code contains a cycle, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/puppetdb-error</td>
<td>If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/query-error</td>
<td>If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

**Related information**

- [Forming orchestrator API requests](#) on page 696

Instructions on interacting with this API.

- [Orchestrator API error responses](#) on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**Tasks in PE**

Tasks are ad-hoc actions you can execute on a target and run from the command line or the console.

A task is a single action that you execute on target machines. With tasks, you can troubleshoot and deploy changes to individual or multiple systems in your infrastructure.

You can run tasks from your tool of choice: the console, the orchestrator command line interface (CLI), or the orchestrator API `/command/task` endpoint.

When you run a task, you can run it immediately, schedule it to run later, or schedule it to run at a recurring frequency - hourly, daily, weekly, every 2 weeks, or every four weeks. After you launch a task, you can check on the status or view the output later with the console or CLI.
If a reboot occurs or if you need to restore a backup, scheduled tasks are rescheduled based on the last execution time. If the scheduled task running in the orchestrator is what caused the reboot, the task run appears as failed.

**Note:** If you are running multiple tasks, make sure your concurrency limit for tasks and bolt-server can accommodate your needs. See Configure the orchestrator and pe-orchestration-services on page 303 to adjust these settings.

- Installing tasks on page 634
  Puppet Enterprise comes with some pre-installed tasks, and you can install or write other tasks you want to use.
- Running tasks in PE on page 634
  Use the orchestrator to set up jobs in the console or on the command line and run Bolt tasks across systems in your infrastructure.
- Writing tasks on page 645
  Bolt tasks are similar to scripts, but they are kept in modules and can have metadata. This allows you to reuse and share them.

### Installing tasks

Puppet Enterprise comes with some pre-installed tasks, and you can install or write other tasks you want to use.

PE includes the following pre-installed tasks:

- **package** - Inspect, install, upgrade, and manage packages.
- **service** - Start, stop, restart, and check the status of a service.
- **facter_task** - Inspect the value of system facts.
- **puppet_conf** - Inspect Puppet agent configuration settings.

You can find other tasks packaged in Puppet modules that you can install from the Forge and then manage with a Puppetfile and Code Manager. To install a module (so that you can use the tasks in the module), select the desired install method under **Start using this module** on the module's Forge page and follow the presented instructions.

You can also write tasks.

### Related information

- Managing environment content with a Puppetfile on page 768
  A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

### Running tasks in PE

Use the orchestrator to set up jobs in the console or on the command line and run Bolt tasks across systems in your infrastructure.

Running a task does not update your Puppet configuration. If you run a task that changes the state of a resource that Puppet is managing, a subsequent Puppet run changes the state of that resource back to what is defined in your Puppet configuration. For example, if you use a task to update the version of a managed package, the version of that package is reset to whatever is specified in a manifest on the next Puppet run.

**Note:** If you have set up compilers and you want to use tasks, you must either set `primary_uris` or `server_list` on agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.
• **Running tasks from the console** on page 635
  Run ad-hoc tasks on target machines to upgrade packages, restart services, or perform any other type of single-action executions on your nodes.

• **Running tasks from the command line** on page 641
  Use the `puppet task run` command to run tasks on agent nodes.

• **Stop a task in progress** on page 645
  You can stop a task in progress if, for example, you realize you need to adjust your PQL query or edit the parameters the task run needs.

• **Inspecting tasks** on page 645
  View the tasks that you have installed and have permission to run, as well as the documentation for those tasks.

**Related information**

Configure compilers on page 248
Compilers must be configured to appropriately route communication between your primary server and agent nodes.

**Running tasks from the console**
Run ad-hoc tasks on target machines to upgrade packages, restart services, or perform any other type of single-action executions on your nodes.

When you set up a job to run a task from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs the tasks on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

There are three ways to specify the job target (the nodes you want to run tasks on):

• A static node list
• A Puppet Query Language (PQL) query
• A node group

You can't combine these methods, and if you switch from one to the other, the target list clears and starts over. In other words, if you create a target list by using a node list, switching to a PQL query clears the node list. You can do a one-time conversion of a PQL query to a static node list if you want to add or remove nodes from the query results.

**Run a task on a node list**
Create a list of target nodes when you need to run a task on a specific set of nodes that isn't easily defined by a PQL query.

**Before you begin**
Install the tasks you want to use.
Make sure you have permission to run the tasks.
Make sure you have access to the nodes you want to target.

**Tip:** You can add network devices to a node list when you have installed modules for device transports in your production environment. You can find such modules in Puppet Forge.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.
4. In the **Task** field, select a task to run, for example **service**.

**Note:** If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

   **Important:** You must click **Add parameter** for each optional parameter-value pair you add to the task.

   To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

   Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

   Tasks with default values run using the default value unless another value is specified.

   **Note:** The parameters you supplied the first time you ran a task will be used for subsequent tasks run when using the **Run again** feature on the **Task details** page.

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

8. From the list of target types, select **Node list**.

9. Create the node list.
   a) Expand the **Inventory nodes** target.
   b) Enter the name of the node you want to find and click **Search**.

   **Note:** The search does not handle regular expressions but does support partial matches.

   c) From the list of results, select the nodes that you want to add to your list. They are added to a table below.
   d) Repeat the search to add other nodes. You can select nodes from multiple searches to create the node list.

   **Tip:** To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

    Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

    **Tip:** Filter run results by task name to find specific task runs.

**Run a task over SSH**

Use the SSH protocol to run tasks on target nodes that do not have the Puppet agent installed.

**Before you begin**

Install the tasks you want to use.

Make sure you have permission to run the tasks on all nodes.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to production.
4. In the **Task** field, select a task to run, for example **service**.

   **Note:** If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

**Important:** You must click **Add parameter** for each optional parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

**Note:** The parameters you supplied the first time you ran a task will be used for subsequent tasks run when using the **Run again** feature on the **Task details** page.

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

8. From the list of target types, select **Node list**.

9. Create the node list.
   a) Expand the **SSH nodes** target.

   **Note:** This target is available only for tasks permitted to run on all nodes.

   b) Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.

   c) Optional: Select additional target options.

   For example, to add a target port number, select **Target port** from the drop-down list, enter the number, and click **Add**.

   d) Click **Add nodes**. They are added to a table below.

   e) Repeat these steps to add other nodes. You can add SSH nodes with different credentials to create the node list.

   **Tip:** To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

    Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

    **Tip:** Filter run results by task name to find specific task runs.

**Run a task over WinRM**

Use the Windows Remote Management (WinRM) to run tasks on target nodes that do not have the Puppet agent installed.

**Before you begin**

Install the tasks you want to use.

Make sure you have permission to run the tasks on all nodes.

1. In the console, in the **Orchestration** section, click **Tasks**.

2. Click **Run a task** in the upper right corner of the **Tasks** page.

3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.

4. In the **Task** field, select a task to run, for example service.

   **Note:** If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.
5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

   **Important:** You must click **Add parameter** for each optional parameter-value pair you add to the task.

   To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

   Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

   Tasks with default values run using the default value unless another value is specified.

   **Note:** The parameters you supplied the first time you ran a task will be used for subsequent tasks run when using the **Run again** feature on the **Task details** page.

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

8. From the list of target types, select **Node list**.

9. Create the node list.
   a) Expand the **WinRM nodes** target.

   **Note:** This target is available only for tasks permitted to run on all nodes.

   b) Enter the target host names and the credentials required to access them.

   c) Optional: Select additional target options.

   For example, to add a target port number, select **Target Port** from the drop-down list, enter the number, and click **Add**.

   d) Click **Add nodes**. They are added to a table below.

   e) Repeat these steps to add other nodes. You can add nodes with different credentials to create the node list.

   **Tip:** To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

    Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

    **Tip:** Filter run results by task name to find specific task runs.

---

**Run a task on a PQL query**

Create a PQL query to run tasks on nodes that meet specific conditions.

**Before you begin**

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run tasks and PQL queries.

1. In the console, in the **Orchestration** section, click **Tasks**.

2. Click **Run a task** in the upper right corner of the **Tasks** page.

3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.

---

© 2022 Puppet, Inc., a Perforce company
4. In the **Task** field, select a task to run, for example *service*.

**Note:** If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

**Important:** You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

**Note:** The parameters you supplied the first time you ran a task will be used for subsequent tasks run when using the **Run again** feature on the **Task details** page.

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

8. From the list of target types, select **PQL query**.

9. Specify a target by doing one of the following:

- Enter a query that selects the target you want. See the **Puppet Query Language (PQL) reference** for more information.
- Click **Common queries**. Select one of the queries and replace the defaults in the braces ({} ) with values that specify the target you want.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes</td>
<td>nodes[certname] ( )</td>
</tr>
<tr>
<td>Nodes with a specific resource (example: httpd)</td>
<td>resources[certname] { type = &quot;Service&quot; and title = &quot;httpd&quot; }</td>
</tr>
<tr>
<td>Nodes with a specific fact and value (example: OS name is CentOS)</td>
<td>inventory[certname] { facts.os.name = &quot;CentOS&quot; }</td>
</tr>
<tr>
<td>Nodes with a specific report status (example: last run failed)</td>
<td>reports[certname] { latest_report_status = &quot;failed&quot; }</td>
</tr>
<tr>
<td>Nodes with a specific class (example: Apache)</td>
<td>resources[certname] { type = &quot;Class&quot; and title = &quot;Apache&quot; }</td>
</tr>
<tr>
<td>Nodes assigned to a specific environment (example: production)</td>
<td>nodes[certname] { catalog_environment = &quot;production&quot; }</td>
</tr>
<tr>
<td>Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)</td>
<td>resources[certname] { type = &quot;Package&quot; and title = &quot;openssl&quot; and parameters.ensure = &quot;1.0.1e-51.el7.2.7&quot; }</td>
</tr>
<tr>
<td>Nodes with a specific resource and operating system (example: httpd and CentOS)</td>
<td>inventory[certname] { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }</td>
</tr>
</tbody>
</table>

10. Click **Submit query** and click **Refresh** to update the node results.
11. If you change or edit the query after it runs, click **Submit query** again.

12. Optional: To convert the PQL query target results to a node list, for use as a node list target, click **Convert query to static node list**.

   **Note:** If you select this option, the job target becomes a node list. You can add or remove nodes from the node list before running the job, but you cannot edit the query.

13. Click **Run task** or **Schedule job**.

   Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

   **Tip:** Filter run results by task name to find specific task runs.

**Important:** When you run this job, the PQL query runs again, and the job might run on a different set of nodes than what is currently displayed. If you want the job to run only on the list as displayed, convert the query to a static node list before you run the job.

---

### Add custom PQL queries to the console

Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary sever, copy the `custom_pql_queries.json.example` file and remove the `.example` suffix.

   ```bash
   sudo cp /etc/puppetlabs/console-services/custom_pql_queries.json.example /etc/puppetlabs/console-services/custom_pql_queries.json
   ```

2. Edit the file contents to include your own PQL queries or remove any existing queries.

3. Refresh the console UI in your browser.

   You can now see your custom queries in the PQL drop down options when running jobs.

### Run a task on a node group

Similar to running a task on a list of nodes that you create in the console, you can run a task on a node group.

#### Before you begin

- Install the tasks you want to use.
- Make sure you have permission to run the tasks.
- Make sure you have access to the nodes you want to target.

**Note:** If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

1. In the console, in the **Orchestration** section, click **Tasks**.

2. Click **Run a task** in the upper right corner of the **Tasks** page.

3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.

4. In the **Task** field, select a task to run, for example, **service**.

   **Note:** If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

**Important:** You must click **Add parameter** for each **optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

**Note:** The parameters you supplied the first time you ran a task will be used for subsequent tasks run when using the **Run again** feature on the **Task details** page.

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

8. From the list of target types, select **Node group**.

9. In the **Choose a node group** box, type or select a node group, and click **Select**.

10. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

**Tip:** Filter run results by task name to find specific task runs.

---

**Running tasks from the command line**

Use the `puppet task run` command to run tasks on agent nodes.

Use the `puppet task` tool and the relevant module to make changes arbitrarily, rather than through a Puppet configuration change. For example, to inspect a package or quickly stop a particular service.

You can run tasks on a single node, on nodes identified in a static list, on nodes retrieved by a PQL query, or on nodes in a node group.

Use the orchestrator command `puppet task` to trigger task runs.

The first time you run a command, you need to authenticate. See the [orchestrator installation instructions](#) for information about setting RBAC permissions and token authorization.

**Run a task on a list of nodes or a single node**

Use a node list target when you need to run a job on a set of nodes that doesn't easily resolve to a PQL query. Use a single node or a comma-separated list of nodes.

**Before you begin**

- Install the tasks you want to use.
- Make sure you have access to the nodes you want to target.
- Make sure you have permissions to run tasks and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- **To run a task job on a single node:** `puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --nodes <NODE NAME> <OPTIONS>`
- **To run a task job on a list of nodes,** use a comma-separated list of node names: `puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --nodes <NODE NAME>,<NODE NAME>,<NODE NAME> <OPTIONS>`

**Note:** Do not add spaces in the list of nodes.

---

© 2022 Puppet, Inc., a Perforce company
To run a task job on a node list from a text file:

```bash
puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --nodes @/path/to/file.txt
```

**Note:** In the text file, put each node on a separate line.

For example, to run the service task with two required parameters, on three specific hosts:

```bash
puppet task run service action=status service=nginx --nodes host1,host2,host3
```

**Tip:** Use `puppet task show <TASK NAME>` to see a list of available parameters for a task. Not all tasks require parameters.

Refer to the `puppet task` command options to see how to pass parameters with the `--params` flag.

---

**Run a task on a PQL query**

Create a PQL query to run tasks on nodes that meet specific conditions.

**Before you begin**

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run tasks and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- To specify the query on the command line:
  ```bash
  puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --query '<QUERY>' <OPTIONS>
  ```

- To pass the query in a text file:
  ```bash
  puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --query @/path/to/file.txt
  ```

For example, to run the service task with two required parameters, on nodes with "web" in their certname:

```bash
puppet task run service action=status service=nginx --query 'nodes { certname ~ "web" }'
```

**Tip:** Use `puppet task show <TASK NAME>` to see a list of available parameters for a task. Not all tasks require parameters.

Refer to the `puppet task` command options to see how to pass parameters with the `--params` flag.

The following table shows some example targets and the associated PQL queries you could run with the orchestrator. Be sure to wrap the entire query in single quotes and use double quotes inside the query.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single node by certname</td>
<td>--query 'nodes { certname = &quot;mynode&quot; }'</td>
</tr>
<tr>
<td>All nodes with &quot;web&quot; in certname</td>
<td>--query 'nodes { certname ~ &quot;web&quot; }'</td>
</tr>
<tr>
<td>All CentOS nodes</td>
<td>--query 'inventory { facts.os.name = &quot;CentOS&quot; }'</td>
</tr>
</tbody>
</table>
### Target | PQL query
---|---
All CentOS nodes with httpd managed | `--query 'inventory { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }'`
All nodes with failed reports | `--query 'reports { latest_report? = true and status = "failed" }'`
All nodes matching the environment for the last received catalog | `--query 'nodes { catalog_environment = "production" }'`

**Tip:** You can use `-q` in place of `--query`.

### Run a task on a node group
Similar to running a task on a list of nodes, you can run a task on a node group.

**Before you begin**
Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

1. Log into your primary server or client tools workstation.
2. Run the command: `puppet task run <TASK NAME> --node-group <node-group-id>`

**Tip:** Use the `/v1/groups` endpoint to retrieve a list node groups and their IDs.

### Related information
- **GET /v1/groups** on page 530
  Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

- **Get a list of all node groups and their IDs** on page 549
  Use this curl command to get a list of all node groups and their corresponding IDs. Node group IDs are useful when running tasks or plans.

### Puppet task run command options
The following are common options you can use with the `task` action. For a complete list of global options run `puppet task --help`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--noop</code></td>
<td>Flag, default false</td>
<td>Run a task to simulate changes without actually enforcing the changes.</td>
</tr>
<tr>
<td><code>--params</code></td>
<td>String</td>
<td>Specify a JSON object that includes the parameters, or specify the path to a JSON file containing the parameters, prefaced with @, for example, <code>@/path/to/file.json</code>. Do not use this flag if specifying parameter-value pairs inline; see more information below.</td>
</tr>
<tr>
<td><code>--environment,-e</code></td>
<td>Environment name</td>
<td>Use tasks installed in the specified environment.</td>
</tr>
</tbody>
</table>
You can pass parameters into the task one of two ways:

- **Inline**, using the `<PARAMETER>=<VALUE>` syntax:

  ```
puppet task run <TASK NAME> <PARAMETER>=<VALUE> <PARAMETER>=<VALUE> --nodes <LIST OF NODES>
puppet task run my_task action=status service=my_service timeout=8 --nodes host1,host2,host3
  ```

- **With the --params option**, as a JSON object or reference to a JSON file:

  ```
puppet task run <TASK NAME> --params '{ "action":"status", "service":"my_service", "timeout":8 }' --nodes host1,host2,host3
puppet task run my_task --params @/path/to/file.json --nodes host1,host2,host3
  ```

You can't combine these two ways of passing in parameters; choose either inline or --params. If you use the inline way, parameter types other than string, integer, double, and Boolean will be interpreted as strings. Use the --params method if you want them read as their original type.

**Reviewing task job output**

The output the orchestrator returns depends on the type of task you run. Output is either standard output (STDOUT) or structured output. At minimum, the orchestrator prints a new job ID and the number of nodes in the task.

The following example shows a task to check the status of the Puppet service running on a list of nodes derived from a PQL query.

```
[example@orch-master ~]$ puppet task run service service=puppet action=status -q 'nodes {certname ~ "br"}' --environment=production
Starting job ...
New job ID: 2029
Nodes: 8

  Started on bronze-11 ...
  Started on bronze-8 ...
  Started on bronze-3 ...
  Started on bronze-6 ...
  Started on bronze-2 ...
  Started on bronze-5 ...
  Started on bronze-7 ...
  Started on bronze-10 ...
  Finished on node bronze-11
    status : running
    enabled : true
  Finished on node bronze-3
    status : running
    enabled : true
  Finished on node bronze-8
    status : running
    enabled : true
  Finished on node bronze-7
    status : running
    enabled : true
```
Finished on node bronze-2
status : running
enabled : true
Finished on node bronze-6
status : running
enabled : true
Finished on node bronze-5
status : running
enabled : true
Finished on node bronze-10
status : running
enabled : true

Job completed. 8/8 nodes succeeded.
Duration: 1 sec

Tip: To view the status of all running, completed, and failed jobs run the puppet job show command, or view them from the Job details page in the console.

Stop a task in progress
You can stop a task in progress if, for example, you realize you need to adjust your PQL query or edit the parameters the task run needs.

When you stop a task run, runs that are underway still finish but no new tasks start on the node until you run again.

To stop a task, do one of the following:
• In the console, on the Tasks page, find the task run you want to stop and click Stop job.
• On the command line, press CTRL + C.

Inspecting tasks
View the tasks that you have installed and have permission to run, as well as the documentation for those tasks.

Log into your primary server or client tools workstation and run one of the following commands:
• To check the documentation for a specific task: puppet task show <TASK>. The command returns the following:
  • The command format for running the task
  • Any parameters available to use with the task
• To view a list of your permitted tasks: puppet task show
• To view a list of all installed tasks pass the --all flag: puppet task show --all

Writing tasks
Bolt tasks are similar to scripts, but they are kept in modules and can have metadata. This allows you to reuse and share them.

You can write tasks in any programming language the target nodes run, such as Bash, PowerShell, or Python. A task can even be a compiled binary that runs on the target. Place your task in the ./tasks directory of a module and add a metadata file to describe parameters and configure task behavior.

For a task to run on remote *nix systems, it must include a shebang (#!) line at the top of the file to specify the interpreter.

For example, the Puppet mysql::sql task is written in Ruby and provides the path to the Ruby interpreter. This example also accepts several parameters as JSON on stdin and returns an error.

#!/opt/puppetlabs/puppet/bin/ruby
require 'json'

© 2022 Puppet, Inc., a Perforce company
pe | Orchestrating Puppet runs, tasks, and plans | 646

require 'open3'
require 'puppet'
def get(sql, database, user, password)
cmd = ['mysql', '-e', "#{sql} "]
cmd << "--database=#{database}" unless database.nil?
cmd << "--user=#{user}" unless user.nil?
cmd << "--password=#{password}" unless password.nil?
stdout, stderr, status = Open3.capture3(*cmd) # rubocop:disable Lint/
UselessAssignment
raise Puppet::Error, _("stderr: ' %{stderr}') % { stderr: stderr }") if
status != 0
{ status: stdout.strip }
end
params = JSON.parse(STDIN.read)
database = params['database']
user = params['user']
password = params['password']
sql = params['sql']
begin
result = get(sql, database, user, password)
puts result.to_json
exit 0
rescue Puppet::Error => e
puts({ status: 'failure', error: e.message }.to_json)
exit 1
end
Related information
Task compatibility on page 17
This table shows which version of the Puppet task specification is compatible with each version of PE.
Secure coding practices for tasks
Use secure coding practices when you write tasks and help protect your system.
Note: The information in this topic covers basic coding practices for writing secure tasks. It is not an exhaustive list.
One of the methods attackers use to gain access to your systems is remote code execution, where by running an
allowed script they gain access to other parts of the system and can make arbitrary changes. Because Bolt executes
scripts across your infrastructure, it is important to be aware of certain vulnerabilities, and to code tasks in a way that
guards against remote code execution.
Adding task metadata that validates input is one way to reduce vulnerability. When you require an enumerated
(enum) or other non-string types, you prevent improper data from being entered. An arbitrary string parameter does
not have this assurance.
For example, if your task has a parameter that selects from several operational modes that are passed to a shell
command, instead of
String $mode = 'file'
use
Enum[file,directory,link,socket] $mode = file
If your task has a parameter that identifies a file on disk, ensure that a user can't specify a relative path that takes them
into areas where they shouldn't be. Reject file names that have slashes.

© 2022 Puppet, Inc., a Perforce company


Instead of

```perl
String $path
```

use

```perl
Pattern[/\A[^/\]\]*z/] $path
```

In addition to these task restrictions, different scripting languages each have their own ways to validate user input.

**PowerShell**

In PowerShell, code injection exploits calls that specifically evaluate code. Do not call `Invoke-Expression` or `Add-Type` with user input. These commands evaluate strings as C# code.

Reading sensitive files or overwriting critical files can be less obvious. If you plan to allow users to specify a file name or path, use `Resolve-Path` to verify that the path doesn't go outside the locations you expect the task to access. Use `Split-Path -Parent $path` to check that the resolved path has the desired path as a parent.

For more information, see PowerShell Scripting and Powershell's Security Guiding Principles.

**Bash**

In Bash and other command shells, shell command injection takes advantage of poor shell implementations. Put quotations marks around arguments to prevent the vulnerable shells from evaluating them.

Because the `eval` command evaluates all arguments with string substitution, avoid using it with user input; however you can use `eval` with sufficient quoting to prevent substituted variables from being executed.

Instead of

```bash
eval "echo $input"
```

use

```bash
eval "echo '$input"
```

These are operating system-specific tools to validate file paths: `realpath` or `readlink -f`.

**Python**

In Python malicious code can be introduced through commands like `eval`, `exec`, `os.system`, `os.popen`, and `subprocess.call` with `shell=True`. Use `subprocess.call` with `shell=False` when you include user input in a command or escape variables.

Instead of

```python
os.system('echo '+input)
```

use

```python
subprocess.check_output(['echo', input])
```

Resolve file paths with `os.realpath` and confirm them to be within another path by looping over `os.path.dirname` and comparing to the desired path.

For more information on the vulnerabilities of Python or how to escape variables, see Kevin London's blog post on Dangerous Python Functions.
Ruby

In Ruby, command injection is introduced through commands like eval, exec, system, backtick (`) or %x() execution, or the Open3 module. You can safely call these functions with user input by passing the input as additional arguments instead of a single string.

Instead of

```ruby
system("echo #{flag1} #{flag2}")
```

use

```ruby
system('echo', flag1, flag2)
```

Resolve file paths with Pathname#realpath, and confirm them to be within another path by looping over Pathname#parent and comparing to the desired path.

For more information on securely passing user input, see the blog post Stop using backtick to run shell command in Ruby.

Naming tasks

Task names are named based on the filename of the task, the name of the module, and the path to the task within the module.

You can write tasks in any language that runs on the target nodes. Give task files the extension for the language they are written in (such as .rb for Ruby), and place them in the top level of your module's ./tasks directory.

Task names are composed of one or two name segments, indicating:

- The name of the module where the task is located.
- The name of the task file, without the extension.

For example, the puppetlabs-mysql module has the sql task in ./mysql/tasks/sql.rb, so the task name is mysql::sql. This name is how you refer to the task when you run tasks.

The task filename init is special: the task it defines is referenced using the module name only. For example, in the puppetlabs-service module, the task defined in init.rb is the service task.

Each task or plan name segment:

- Must start with a lowercase letter.
- Can include digits.
- Can include underscores.
- Namespace segments must match the following regular expression: \A[a-z][a-z0-9_]*\Z
- The file extension must not use the reserved extensions .md or .json.

Single-platform tasks

A task can consist of a single executable with or without a corresponding metadata file. For instance, ./mysql/tasks/sql.rb and ./mysql/tasks/sql.json. In this case, no other ./mysql/tasks/sql.* files can exist.

Cross-platform tasks

A task can have multiple implementations, with metadata that explains when to use each one. A primary use case for this is to support different implementations for different target platforms, referred to as cross-platform tasks.

A task can also have multiple implementations, with metadata that explains when to use each one. A primary use case for this is to support different implementations for different target platforms, referred to as cross-platform tasks. For instance, consider a module with the following files:

```
- tasks
  - sql_linux.sh
```
This task has two executables (sql_linux.sh and sql_windows.ps1), each with an implementation metadata file and a task metadata file. The executables have distinct names and are compatible with older task runners such as Puppet Enterprise 2018.1 and earlier. Each implementation has its own metadata which documents how to use the implementation directly or marks it as private to hide it from UI lists.

An implementation metadata example:

```json
{
   "name": "SQL Linux",
   "description": "A task to perform sql operations on linux targets",
   "private": true
}
```

The task metadata file contains an implementations section:

```json
{
   "implementations": [
      {
         "name": "sql_linux.sh",
         "requirements": ["shell"]
      },
      {
         "name": "sql_windows.ps1",
         "requirements": ["powershell"]
      }
   ]
}
```

Each implementation has a name and a list of requirements. The requirements are the set of features which must be available on the target in order for that implementation to be used. In this case, the sql_linux.sh implementation requires the shell feature, and the sql_windows.ps1 implementation requires the PowerShell feature.

The set of features available on the target is determined by the task runner. You can specify additional features for a target via set_feature or by adding features in the inventory. The task runner chooses the first implementation whose requirements are satisfied.

The following features are defined by default:

- **puppet-agent**: Present if the target has the Puppet agent package installed. This feature is automatically added to hosts with the name localhost.
- **shell**: Present if the target has a posix shell.
- **powershell**: Present if the target has PowerShell.

**Sharing executables**

Multiple task implementations can refer to the same executable file.

Executables can access the _task metaparameter, which contains the task name. For example, the following creates the tasks service::stop and service::start, which live in the executable but appear as two separate tasks.

```ruby
#!/usr/bin/env ruby
require 'json'
params = JSON.parse(STDIN.read)
action = params['action'] || params['_task']
if ['start', 'stop'].include?(action)
   `systemctl #{params['_task']} #{params['service']}`
```
myservice/tasks/start.json

{
    "description": "Start a service",
    "parameters": {
        "service": {
            "type": "String",
            "description": "The service to start"
        }
    },
    "implementations": [
        {"name": "init.rb"}
    ]
}

myservice/tasks/stop.json

{
    "description": "Stop a service",
    "parameters": {
        "service": {
            "type": "String",
            "description": "The service to stop"
        }
    },
    "implementations": [
        {"name": "init.rb"}
    ]
}

Sharing task code

Multiple tasks can share common files between them. Tasks can additionally pull library code from other modules.

To create a task that includes additional files pulled from modules, include the files property in your metadata as an array of paths. A path consists of:

- the module name
- one of the following directories within the module:
  - files — Most helper files. This prevents the file from being treated as a task or added to the Puppet Ruby loadpath.
  - tasks — Helper files that can be called as tasks on their own.
  - lib — Ruby code that might be reused by types, providers, or Puppet functions.
- the remaining path to a file or directory; directories must include a trailing slash /

All path separators must be forward slashes. An example would be stdlib/lib/puppet/.

The files property can be included both as a top-level metadata property, and as a property of an implementation, for example:

```json
{
    "implementations": [
        {"name": "sql_linux.sh", "requirements": ["shell"], "files": ["mymodule/files/lib.sh"],
        {"name": "sql_windows.ps1", "requirements": ["powershell"], "files": ["mymodule/files/lib.ps1"]}
    ]
}
```
When a task includes the `files` property, all files listed in the top-level property and in the specific implementation chosen for a target are copied to a temporary directory on that target. The directory structure of the specified files is preserved such that paths specified with the `files` metadata option are available to tasks prefixed with `_installdir`. The task executable itself is located in its module location under the `_installdir` as well, so other files can be found at `../../../mymodule/files/` relative to the task executable's location.

For example, you can create a task and metadata in a module at `~/.puppetlabs/bolt/site-modules/mymodule/tasks/task.{json,rb}.

Metadata

```json
{
  "files": ["multi_task/files/rb_helper.rb"]
}
```

File resource

`multi_task/files/rb_helper.rb`

```ruby
def useful_ruby
  { helper: "ruby" }
end
```

Task

```ruby
#!/usr/bin/env ruby
require 'json'

params = JSON.parse(STDIN.read)
require_relative File.join(params['_installdir'], 'multi_task', 'files', 'rb_helper.rb')
# Alternatively use relative path
# require_relative File.join(__dir__, '..', '..', 'multi_task', 'files', 'rb_helper.rb')
puts useful_ruby.to_json
```

Output

```bash
Started on localhost...
Finished on localhost:

  "helper": "ruby"

Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

Task helpers

To help with writing tasks, Bolt includes `python_task_helper` and `ruby_task_helper`. It also makes a useful demonstration of including code from another module.

Python example

Create task and metadata in a module at `~/.puppetlabs/bolt/site-modules/mymodule/tasks/task.{json,py}.`
### Metadata

```json
{
  "files": ["python_task_helper/files/task_helper.py"],
  "input_method": "stdin"
}
```

### Task

```python
#!/usr/bin/env python
import os, sys
sys.path.append(os.path.join(os.path.dirname(__file__), '..', '..','python_task_helper', 'files'))
from task_helper import TaskHelper

class MyTask(TaskHelper):
    def task(self, args):
        return {'greeting': 'Hi, my name is ' + args['name']}

if __name__ == '__main__':
    MyTask().run()
```

### Output

```
$ bolt task run mymodule::task -n localhost name='Julia'
Started on localhost...
Finished on localhost:
  {
    "greeting": "Hi, my name is Julia"
  }
Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

### Ruby example

Create task and metadata in a new module at ~/.puppetlabs/bolt/site-modules/mymodule/tasks/mytask.(json,rb).

### Metadata

```json
{
  "files": ["ruby_task_helper/files/task_helper.rb"],
  "input_method": "stdin"
}
```

### Task

```ruby
#!/usr/bin/env ruby
require_relative '../../ruby_task_helper/files/task_helper.rb'
class MyTask < TaskHelper
    def task(name: nil, **kwargs)
        { greeting: "Hi, my name is #{name}" }
    end
end

MyTask.run if __FILE__ == $0
```
Output

```bash
$ bolt task run mymodule::mytask -n localhost name="Robert'"); DROP TABLE Students;--"
Started on localhost...
Finished on localhost:
  { "greeting": "Hi, my name is Robert'"); DROP TABLE Students;--"
  }
Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

Writing remote tasks

Some targets are hard or impossible to execute tasks on directly. In these cases, you can write a task that runs on a proxy target and remotely interacts with the real target.

For example, a network device might have a limited shell environment or a cloud service might be driven only by HTTP APIs. By writing a remote task, Bolt allows you to specify connection information for remote targets in their inventory file and injects them into the _target metaparam.

This example shows how to write a task that posts messages to Slack and reads connection information from inventory.yaml:

```ruby
#!/usr/bin/env ruby
# modules/slack/tasks/message.rb
require 'json'
require 'net/http'

params = JSON.parse(STDIN.read)
# the slack API token is passed in from inventory
token = params['_target']['token']

uri = URI('https://slack.com/api/chat.postMessage')
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true

req = Net::HTTP::Post.new(uri, 'Content-type' => 'application/json')
req['Authorization'] = "Bearer #{params['_target']['token']}"
req.body = { channel: params['channel'], text: params['message'] }.to_json

resp = http.request(req)
puts resp.body
```

To prevent accidentally running a normal task on a remote target and breaking its configuration, Bolt won't run a task on a remote target unless its metadata defines it as remote:

```yaml
{
  "remote": true
}
```

Add Slack as a remote target in your inventory file:

```yaml
---
nodes:
- name: my_slack
  config:
    transport: remote
    remote:
Finally, make my_slack a target that can run the slack::message:

```bash
bolt task run slack::message --nodes my_slack message="hello" channel=<slack channel id>
```

### Defining parameters in tasks

Allow your task to accept parameters as either environment variables or as a JSON hash on standard input.

Tasks can receive input as either environment variables, a JSON hash on standard input, or as PowerShell arguments. By default, the task runner submits parameters as both environment variables and as JSON on stdin.

If your task needs to receive parameters only in a certain way, such as stdin only, you can set the input method in your task metadata. For Windows tasks, it's usually better to use tasks written in PowerShell. See the related topic about task metadata for information about setting the input method.

Environment variables are the easiest way to implement parameters, and they work well for simple JSON types such as strings and numbers. For arrays and hashes, use structured input instead because parameters with undefined values (nil, undef) passed as environment variables have the String value null. For more information, see Structured input and output on page 656.

To add parameters to your task as environment variables, pass the argument prefixed with the Puppet task prefix PT_.

For example, to add a message parameter to your task, read it from the environment in task code as PT_message. When the user runs the task, they can specify the value for the parameter on the command line as `message=hello`, and the task runner submits the value hello to the PT_message variable.

```bash
#!/usr/bin/env bash
echo your message is $PT_message
```

### Defining parameters in Windows

For Windows tasks, you can pass parameters as environment variables, but it's easier to write your task in PowerShell and use named arguments. By default tasks with a .ps1 extension use PowerShell standard argument handling.

For example, this PowerShell task takes a process name as an argument and returns information about the process. If no parameter is passed by the user, the task returns all of the processes.

```powershell
[CmdletBinding()]
Param(
    [Parameter(Mandatory = $False)]
    [String]
    $Name
)
if ($Name -eq $null -or $Name -eq "") {
    Get-Process
} else {
    $processes = Get-Process -Name $Name
    $result = @()
    foreach ($process in $processes) {
        $result += @{
            "Name" = $process.ProcessName;
            "CPU" = $process.CPU;
            "Memory" = $process.WorkingSet;
            "Path" = $process.Path;
            "Id" = $process.Id
        }
    }
    if ($result.Count -eq 1) {
        ConvertTo-Json -InputObject $result[0] -Compress
    }
```
To pass parameters in your task as environment variables (PT_parameter), you must set input_method in your task metadata to environment. To run Ruby tasks on Windows, the Puppet agent must be installed on the target nodes.

**Defining sensitive parameters**

You can define task parameters, like passwords or API keys, as sensitive. The parameter is then masked when it appears in logs and API responses. When you want to view these values, set the log file to the correct level based on the service.

To define a parameter as sensitive within the JSON metadata, add the "sensitive": true property.

```
{
  "description": "This task has a sensitive property denoted by its metadata",
  "input_method": "stdin",
  "parameters": {
    "user": {
      "description": "The user",
      "type": "String[1]"
    },
    "password": {
      "description": "The password",
      "type": "String[1]",
      "sensitive": true
    }
  }
}
```

Some services log sensitive parameter values. Here are the minimum log levels for each service where sensitive values can be seen.

<table>
<thead>
<tr>
<th>Service</th>
<th>Minimum log level that shows sensitive parameters</th>
<th>Default log level</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-bolt-server</td>
<td>INFO</td>
<td>INFO</td>
</tr>
<tr>
<td>pe-ace-server</td>
<td>INFO</td>
<td>INFO</td>
</tr>
<tr>
<td>pxp-agent</td>
<td>DEBUG</td>
<td>INFO</td>
</tr>
<tr>
<td>pcp-broker</td>
<td>TRACE</td>
<td>INFO</td>
</tr>
</tbody>
</table>

If you don't want sensitive parameters to be logged, configure the relevant service's log level to be one level higher than its minimum sensitive parameter value log level. For example, to avoid seeing sensitive parameters for pxp-agent, set the log level to INFO, which is one level higher than DEBUG, the minimum log level that shows sensitive parameters for pxp-agent.

**Returning errors in tasks**

To return a detailed error message if your task fails, include an Error object in the task's result.

When a task exits non-zero, the task runner checks for an error key ('_error'). If one is not present, the task runner generates a generic error and adds it to the result. If there is no text on stdout but text is present on stderr, the stderr text is included in the message.

```
{"_error": {
  "msg": "Task exited 1:
Something on stderr",
  "kind": "puppetlabs.tasks/task-error",
}}
```

© 2022 Puppet, Inc., a Perforce company
An error object includes the following keys:

**msg**
A human readable string that appears in the UI.

**kind**
A standard string for machines to handle. You can share kinds between your modules or namespace kinds per module.

**details**
An object of structured data about the tasks.

Tasks can provide more details about the failure by including their own error object in the result at _error.

```ruby
#!/opt/puppetlabs/puppet/bin/ruby
require 'json'
begin
  params = JSON.parse(STDIN.read)
  result = {}
  result['result'] = params['dividend'] / params['divisor']
rescue ZeroDivisionError
  result[:_error] = { msg: "Cannot divide by zero",
                     kind: "puppetlabs-example_modules/dividebyzero",
                     details: { divisor: divisor },
                   }
rescue Exception => e
  result[:_error] = { msg: e.message,
                     kind: "puppetlabs-example_modules/unknown",
                     details: { class: e.class.to_s },
                   }
end
puts result.to_json
```

**Structured input and output**

If you have a task that has many options, returns a lot of information, or is part of a task plan, consider using structured input and output with your task.

The task API is based on JSON. Task parameters are encoded in JSON, and the task runner attempts to parse the output of the tasks as a JSON object.

The task runner can inject keys into that object, prefixed with __. If the task does not return a JSON object, the task runner creates one and places the output in an _output_ key.

**Structured input**

For complex input, such as hashes and arrays, you can accept structured JSON in your task.

By default, the task runner passes task parameters as both environment variables and as a single JSON object on stdin. The JSON input allows the task to accept complex data structures.

To accept parameters as JSON on stdin, set the params key to accept JSON on stdin.

```ruby
#!/opt/puppetlabs/puppet/bin/ruby
require 'json'
```
params = JSON.parse(STDIN.read)
exitcode = 0
params['files'].each do |filename|
    begin
        FileUtils.touch(filename)
        puts "updated file #{filename}"
    rescue
        exitcode = 1
        puts "couldn't update file #{filename}"
    end
end
exit exitcode

If your task accepts input on stdin it should specify "input_method": "stdin" in its metadata.json file, or it might not work with sudo for some users.

**Returning structured output**
To return structured data from your task, print only a single JSON object to stdout in your task.

Structured output is useful if you want to use the output in another program, or if you want to use the result of the task in a Puppet task plan.

```python
#!/usr/bin/env python
import json
import sys
minor = sys.version_info
result = { "major": sys.version_info.major, "minor": sys.version_info.minor }
json.dump(result, sys.stdout)
```

**Converting scripts to tasks**
To convert an existing script to a task, you can either write a task that wraps the script or you can add logic in your script to check for parameters in environment variables.

If the script is already installed on the target nodes, you can write a task that wraps the script. In the task, read the script arguments as task parameters and call the script, passing the parameters as the arguments.

If the script isn't installed or you want to make it into a cohesive task so that you can manage its version with code management tools, add code to your script to check for the environment variables, prefixed with PT_, and read them instead of arguments.

⚠️ **CAUTION:** For any tasks that you intend to use with PE and assign RBAC permissions, make sure the script safely handles parameters or validate them to prevent shell injection vulnerabilities.

Given a script that accepts positional arguments on the command line:

```bash
version=$1
[ -z "$version" ] && echo "Must specify a version to deploy && exit 1
if [ -z "$2" ]; then
    filename=$2
else
    filename=~/.myfile
fi
```

To convert the script into a task, replace this logic with task variables:

```bash
version=$PT_version #no need to validate if we use metadata
if [ -z "$PT_filename" ]; then
```
filename=$PT_filename
else
    filename=~/.myfile
fi

Wrapping an existing script
If a script is not already installed on targets and you don't want to edit it, for example if it's a script someone else maintains, you can wrap the script in a small task without modifying it.

⚠️ CAUTION: For any tasks that you intend to use with PE and assign RBAC permissions, make sure the script safely handles parameters or validate them to prevent shell injection vulnerabilities.

Given a script, myscript.sh, that accepts 2 positional args, filename and version:

1. Copy the script to the module's files/ directory.
2. Create a metadata file for the task that includes the parameters and file dependency.

```json
{
    "input_method": "environment",
    "parameters": {
        "filename": { "type": "String[1]" },
        "version": { "type": "String[1]" }
    },
    "files": [
        "script_example/files/myscript.sh"
    ]
}
```

3. Create a small wrapper task that reads environment variables and calls the task.

```bash
#!/usr/bin/env bash
set -e
script_file="$PT__installdir/script_example/files/myscript.sh"
# If this task is going to be run from windows nodes
# the wrapper must make sure it's executable
chmod +x $script_file
commandline=($script_file "$PT_filename" "$PT_version")
# If the stderr output of the script is important redirect it to stdout.
"${commandline[@]}" 2>&1
```

Supporting no-op in tasks
Tasks support no-operation functionality, also known as no-op mode. This function shows what changes the task would make, without actually making those changes.

No-op support allows a user to pass the --noop flag with a command to test whether the task will succeed on all targets before making changes.

To support no-op, your task must include code that looks for the _noop metaparameter. No-op is supported only in Puppet Enterprise.

If the user passes the --noop flag with their command, this parameter is set to true, and your task must not make changes. You must also set supports_noop to true in your task metadata or the task runner will refuse to run the task in no-op mode.

No-op metadata example

```json
{
    "description": "Write content to a file.",
    "supports_noop": true,
    "parameters": {
        "filename": {
            "description": "the file to write to",
            "type": "String[1]"
        },
        "content": {
            "description": "The content to write",
            "type": "String"
        }
    }
}
```
No-op task example

```python
#!/usr/bin/env python
import json
import os
import sys

params = json.load(sys.stdin)
filename = params['filename']
content = params['content']
noop = params.get('_noop', False)

exitcode = 0

def make_error(msg):
    error = {
        "_error": {
            "kind": "file_error",
            "msg": msg,
            "details": {}
        }
    }
    return error

try:
    if noop:
        path = os.path.abspath(os.path.join(filename, os.pardir))
        file_exists = os.access(filename, os.F_OK)
        file_writable = os.access(filename, os.W_OK)
        path_writable = os.access(path, os.W_OK)

        if path_writable == False:
            exitcode = 1
            result = make_error("Path %s is not writable" % path)
        elif file_exists == True and file_writable == False:
            exitcode = 1
            result = make_error("File %s is not writable" % filename)
        else:
            result = { "success": True , '_noop': True }
    else:
        with open(filename, 'w') as fh:
            fh.write(content)
        result = { "success": True }
except Exception as e:
    exitcode = 1
    result = make_error("Could not open file %s: %s" % (filename, str(e)))
print(json.dumps(result))
exit(exitcode)
```

Task metadata

Task metadata files describe task parameters, validate input, and control how the task runner executes the task.

Your task must have metadata to be published and shared on the Forge. Specify task metadata in a JSON file with the naming convention `<TASKNAME>.json`. Place this file in the module's ./tasks folder along with your task file.

For example, the module puppetlabs-mysql includes the mysql::sql task with the metadata file, sql.json.
Adding parameters to metadata

To document and validate task parameters, add the parameters to the task metadata as JSON object, `parameters`.

If a task includes `parameters` in its metadata, the task runner rejects any parameters input to the task that aren't defined in the metadata.

In the `parameter` object, give each parameter a description and specify its Puppet type. For a complete list of types, see the types documentation.

For example, the following code in a metadata file describes a `provider` parameter:

```json
"provider": {
    "description": "The provider to use to manage or inspect the service, defaults to the system service manager",
    "type": "Optional[String[1]]"
}
```

Define default parameters

You can define default task parameters, which are supplied to a task run even if the user does not specify a value for the parameter.

For example, the default location for this `log_location` parameter is `/var/log/puppetlabs`

```json
"log_location": {
    "type": "String",
    "description": "The location the log will be stored in",
    "default": "/var/log/puppetlabs"
}
```

Note: Parameters with defaults are considered optional.

Define sensitive parameters

You can define task parameters as sensitive, for example, passwords and API keys. These values are masked when they appear in logs and API responses. When you want to view these values, set the log file to `level: debug`. 

© 2022 Puppet, Inc., a Perforce company
To define a parameter as sensitive within the JSON metadata, add the "sensitive": true property.

```
{
  "description": "This task has a sensitive property denoted by its metadata",
  "input_method": "stdin",
  "parameters": {
    "user": {
      "description": "The user",
      "type": "String[1]"
    },
    "password": {
      "description": "The password",
      "type": "String[1]",
      "sensitive": true
    }
  }
}
```

**Task metadata reference**

The following table shows task metadata keys, values, and default values.

### Table 1: Task metadata

<table>
<thead>
<tr>
<th>Metadata key</th>
<th>Description</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;description&quot;</td>
<td>A description of what the task does.</td>
<td>String</td>
<td>None</td>
</tr>
<tr>
<td>&quot;input_method&quot;</td>
<td>What input method the task runner should use to pass parameters to the task.</td>
<td>Array of objects describing each parameter</td>
<td>Both environment and stdin unless .ps1 tasks, in which case powershell</td>
</tr>
<tr>
<td>&quot;parameters&quot;</td>
<td>The parameters or input the task accepts listed with a puppet type string and optional description. See adding parameters to metadata for usage information.</td>
<td>Integer</td>
<td>1 (This is the only valid value.)</td>
</tr>
<tr>
<td>&quot;puppet_task_version&quot;</td>
<td>The version of the spec used.</td>
<td>Array of Objects describing each implementation</td>
<td>None</td>
</tr>
<tr>
<td>&quot;supports_noop&quot;</td>
<td>Whether the task supports no-op mode. Required for the task to accept the --noop option on the command line.</td>
<td>Array of Objects describing each implementation</td>
<td>None</td>
</tr>
<tr>
<td>&quot;implementations&quot;</td>
<td>A list of task implementations and the requirements used to select one to run. See Cross-platform tasks on page 648 for usage information.</td>
<td>Array of Objects describing each implementation</td>
<td>None</td>
</tr>
<tr>
<td>Metadata key</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>&quot;files&quot;</td>
<td>A list of files to be provided when running the task, addressed by module. See <a href="#">Sharing task code</a> on page 650 for usage information.</td>
<td>Array of Strings</td>
<td>None</td>
</tr>
</tbody>
</table>

### Task metadata types
Task metadata can accept most Puppet data types.

**Restriction:**
Some types supported by Puppet can not be represented as JSON, such as `Hash[Integer, String], Object, or Resource`. These should not be used in tasks, because they can never be matched.

### Table 2: Common task data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Accepts any string.</td>
</tr>
<tr>
<td>String[1]</td>
<td>Accepts any non-empty string (a String of at least length 1).</td>
</tr>
<tr>
<td>Enum[choice1, choice2]</td>
<td>Accepts one of the listed choices.</td>
</tr>
<tr>
<td>Pattern[\A\w+\Z/]</td>
<td>Accepts Strings matching the regex /\w+/ or non-empty strings of word characters.</td>
</tr>
<tr>
<td>Integer</td>
<td>Accepts integer values. JSON has no Integer type so this can vary depending on input.</td>
</tr>
<tr>
<td>Optional[String[1]]</td>
<td>Optional makes the parameter optional and permits null values. Tasks have no required nullable values.</td>
</tr>
<tr>
<td>Array[String]</td>
<td>Matches an array of strings.</td>
</tr>
<tr>
<td>Hash</td>
<td>Matches a JSON object.</td>
</tr>
<tr>
<td>Variant[Integer, Pattern[\A\d+\Z/]]</td>
<td>Matches an integer or a String of an integer</td>
</tr>
<tr>
<td>Boolean</td>
<td>Accepts Boolean values.</td>
</tr>
</tbody>
</table>

### Specifying parameters
Parameters for tasks can be passed to the `bolt` command as CLI arguments or as a JSON hash.

To pass parameters individually to your task or plan, specify the parameter value on the command line in the format `parameter=value`. Pass multiple parameters as a space-separated list. Bolt attempts to parse each parameter value as JSON and compares that to the parameter type specified by the task or plan. If the parsed value matches the type, it is used; otherwise, the original string is used.

For example, to run the `mysql::sql` task to show tables from a database called `mydatabase`:

```bash
bolt task run mysql::sql database=mydatabase sql="SHOW TABLES" --nodes neptune --modules ~/modules
```

To pass a string value that is valid JSON to a parameter that would accept both quote the string. For example to pass the string `true` to a parameter of type `Variant[String, Boolean]` use `'foo="true"'`. To pass a String
value wrapped in "quote and escape it 'string="\"val\"'. Alternatively, you can specify parameters as a single JSON object with the --params flag, passing either a JSON object or a path to a parameter file.

To specify parameters as JSON, use the parameters flag followed by the JSON: --params '{"name": "openssl"}'

To set parameters in a file, specify parameters in JSON format in a file, such as params.json. For example, create a params.json file that contains the following JSON:

```json
{
    "name": "openssl"
}
```

Then specify the path to that file (starting with an at symbol, @) on the command line with the parameters flag: --params @params.json

---

### Plans in PE

Plans allow you to tie together tasks, scripts, commands, and other plans to create complex workflows with refined access control. You can install modules that contain plans or write your own, then run them from the console or the command line.

A plan is a bundle of tasks, commands, scripts, or other plans that can be combined with other logic. They allow you to do complex operations, like running multiple tasks with one command or running certain tasks based on the output of another task.

You can run plans using the tool of your choice: the console, the command line, or the orchestrator API POST /command/plan_run on page 712 endpoint.

RBAC for plans and tasks do not intersect. This means that if a user does not have access to a specific task but they have access to run a plan containing that task, they are still able to run the plan. This allows you to implement more customized access control to tasks by wrapping them within plans. See Defining plan permissions on page 671 for information about RBAC considerations when writing plans or managing plan access.

---

**Note:**

If you have set up compilers and you want to use plans, you must set either primary_uris or server_list on your agents to point to your compilers.

If you are running multiple tasks or tasks within plans, make sure the task_concurrency and Bolt server concurrency limits can support the number of tasks you need to run simultaneously. To adjust these settings, refer to Configure the orchestrator and pe-orchestration-services on page 303.
• **Plans in PE versus Bolt plans** on page 664
Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

• **Installing plans** on page 666
Plans are packaged in modules and deployed with Code Manager. PE includes some pre-installed plans. You can also download modules that contain plans from the Forge and write custom plans.

• **Running plans in PE** on page 667
The orchestrator can run plans across systems in your infrastructure. You can set up plan jobs from the Puppet Enterprise (PE) console or the command line. Plan jobs can run once or on a recurring schedule.

• **Writing plans** on page 670
Plans allow you to run more than one task with a single command, compute values for the input to a task, and run other complex workflows at the same time. They also allow you greater flexibility for creating custom RBAC limitations by wrapping tasks and other workflows within plans.

---

### Plans in PE versus Bolt plans

Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

#### Unavailable plan language functions

The following Bolt plan functions don't work in PE because they haven't been implemented yet or cause issues during plan runs:

- add_to_group
- background
- dir::children
- download_file
- file::exists
- file::readable
- file::write
- get_resources
- out::verbose
- parallelize
- prompt
- prompt::menu
- remove_from_group
- resolve_references
- resource
- run_task_with
- set_config
- set_feature
- set_resources
- set_var
- system::env
- wait
- write_file

#### Apply blocks

The apply feature, including apply_prep, only works for targets using the PXP agent and the PCP transport. It fails on remote devices and on targets connected via SSH or WinRM.
Target groups

Support for target groups is unavailable in PE. Using `add_to_group` causes a plan to fail and referencing a group name in `get_targets` doesn't return any nodes. When using `get_targets` you must reference either node certnames or supply a PuppetDB query. Here is an example of a plan using `get_targets` with node certnames:

```perl
plan example::get_targets_example () {
  $nodes = get_targets(['node1.example.com', 'node2.example.com'])
  run_command('whoami', $nodes)
}
```

Target behaviors

PE assumes all target references can be matched to one of the following:

- A connected agent with a certname
- An entry in the PE inventory service

Therefore, target names must match either a certname or an entry in the PE inventory service.

New targets cannot be added to the inventory service inside a plan. New target objects created in plans can't connect because PE doesn't recognize them.

Targets return an empty hash when asked about connection information.

Target configuration

While you can set up node transport configuration through the PE inventory for nodes to use SSH or WinRM, you can't change the configuration settings for targets from within a plan. Using the `set_config` function in a plan causes the plan to fail and referencing a target object’s configuration hash always returns an empty hash.

The use of URIs in a target name to override the transport is also not supported. All references to targets (i.e. when using `get_targets`) must be either PuppetDB queries or valid certnames that are already in the PE inventory.

Here is an example of a plan that uses `get_targets` correctly:

```perl
plan example::get_targets_example () {
  ## NOTE! If you used ssh://node1.example.com as the first entry, this plan would fail!
  $nodes = get_targets(['node1.example.com', 'node2.example.com'])
  run_command('whoami', $nodes)
}
```

The localhost target

The special target `localhost` is not available for plans in PE. Using `localhost` anywhere in a plan results in a plan failure. If you need to run a plan on the primary server host, use the primary server's certname to reference it.

For example, you can use the following plan for the primary server host `my-primary-server.company.com`:

```perl
plan example::referencing_the_primary_server(){
  # Note that if you tried to use `localhost` instead of `my-primary-server` this plan would fail!
  run_command('whoami', 'my-primary-server.company.com')
}
```

The _run_as parameter

Plans in PE do not support the `_run_as` parameter for changing the user that accesses hosts or executes actions. If this parameter is supplied to any plan function, the plan runs but the user doesn't change.
For example, the following plan is valid, but can't run as other_user:

```puppet
plan example::run_as_example (TargetSpec $nodes) {
  run_command('whoami', $nodes, _run_as => 'other_user')
}
```

**Script and file sources**

When using `run_script` or `file::read`, the source location for the files **must** be from a module that uses a `modulename/filename` selector for a file or directory in `$MODULEROOT/files`. PE doesn't support file sources that reference absolute paths.

The following two code examples show a module structure and plan that correctly use the `modulename/filename` selector:

```plaintext
example/
  ### files
  ###my_script.sh
  ### plans
  ###run_script_example.pp

plan example::run_script_example (TargetSpec $nodes) {
  run_script('example/my_script.sh', $nodes)
}
```

**Code deployment for plans**

Using plans in PE requires [Managing code with Code Manager](#) on page 774. You must enable Code Manager to deploy code to your primary server.

Primary servers deploy a second `codedir` from which plans load code. This secondary code location on your primary server impacts standard module functionality:

- You can't use the `puppet module install` command to install modules for plans, because the `puppet module tool` can't install to the plan `codedir`. However, the `puppet module install` command works as usual for non-plan Puppet code executed and compiled from Puppet Server.
- A `$modulepath` configuration that uses fully qualified paths might not work for plans if they reference the standard `/etc/puppetlabs/code` location. It is more reliable to use relative paths in `$modulepath`.

**Installing plans**

Plans are packaged in modules and deployed with Code Manager. PE includes some pre-installed plans. You can also download modules that contain plans from the Forge and write custom plans.

**Important:** Built-in plans can work without Code Manager; however, you must use Code Manager to install custom plans and plans from the Forge.

To install a new module containing a plan:

1. Find the module you want on the Forge.
2. Under **Start using this module**, select **r10k or Code Manager** as the **Installation method**.

In the Forge, modules containing plans have a **Plans** section in the README.

You can also learn how to write plans in Puppet language and check out some example plans.
Running plans in PE

The orchestrator can run plans across systems in your infrastructure. You can set up plan jobs from the Puppet Enterprise (PE) console or the command line. Plan jobs can run once or on a recurring schedule.

Plans can't change your Puppet configuration, which defines the desired state of your infrastructure. If you run a plan that changes the state of a resource managed by Puppet (for example, upgrading a package or service), the next Puppet run changes that resource's state back to what is defined in your Puppet configuration.

If you have set up compilers and you want to use plans, you must set either `primary_uris` or `server_list` on your agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.

**Important:**

Plans in PE require Managing code with Code Manager on page 774.

Unlike Tasks in PE on page 633, you cannot stop a plan that is in progress.

- Running plans from the console on page 667
  - Run ad hoc plans from the console.
- Running plans from the command line on page 668
  - Run a plan using the `puppet plan run` command.
- Inspecting plans on page 669
  - View the plans you have installed and which ones you have permissions to run, as well as individual plan metadata.
- Running plans alongside code deployments on page 669
  - The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

**Running plans from the console**

Run ad hoc plans from the console.

**Before you begin**

Install the plans you want to use.

Make sure you have the Plans permission to run the plans on the nodes.

When you set up a plan run from the console, the orchestrator creates an ID to track the plan run, shows the nodes included in the plan, and runs the plan on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the plan run.

1. In the console, in the **Orchestration** section, select **Plans** and then click **Run a plan**.
2. Under **Code environment**, select the environment where you installed the module containing the plan you want to run. For example, **production**.
3. Optional: Under **Job description**, provide a description. This text appears on the **Plans** page.
4. Under **Plan**, select the plan you want to run.
5. Under **Plan parameters**, add optional parameters, then enter values for the optional and required parameters on the list. Click **Add parameter** for each optional parameter-value pair you add to the plan.
   - To view information about required and optional parameters for the plan, select **view plan metadata** below the **Plan** field.
   - Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.
   - Plans with default values run using the default unless you specify another value.
6. Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the plan to run.
7. Select **Run job**.

Your plan status and output appear on the **Plans** page.

**Running plans from the command line**

Run a plan using the `puppet plan run` command.

On the command line, run the command `puppet plan run` with the following information included:

- The full name of the plan, formatted as `<MODULE>::<PLAN>`.
- Any plan parameters.

**Note:** To find out what parameters can be included in a plan, view the plan metadata by running the command `puppet plan show <PLAN NAME>` on the command line. For more information, see **Inspecting plans** on page 669.

- Credentials, if required, formatted with the `--user` and `--password` flags.

For example, if a plan defined in `mymodule/plans/myplan.pp` accepts a `load_balancer` parameter, run:

```
puppet plan run mymodule::myplan load_balancer=lb.myorg.com
```

You can pass a comma-separated list of node names, wildcard patterns, or group IDs to a plan parameter that is passed to a run function or that the plan resolves using `get_targets`.

**Related information**

*Get a list of all node groups and their IDs* on page 549

Use this curl command to get a list of all node groups and their corresponding IDs. Node group IDs are useful when running tasks or plans.

**Plan command options**

The following are common options you can use with the `plan` action. For a complete list of global options run `puppet plan --help`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--params</code></td>
<td>A string value used to specify either a JSON object that includes the parameters or the path to a JSON file containing the parameters, prefaced with @. For example, @/path/to/file.json. Do not use this flag if specifying inline parameter-value pairs.</td>
</tr>
<tr>
<td><code>--environment</code> or <code>-e</code></td>
<td>The name of the environment where the plan is installed.</td>
</tr>
<tr>
<td><code>--description</code></td>
<td>A flag used to provide a description for the job to be shown on the job list and job details pages and returned with the <code>puppet job show</code> command. It defaults to empty.</td>
</tr>
</tbody>
</table>

You can pass parameters into the plan one of two ways:

- Inline, using the `<PARAMETER>=<VALUE>` syntax.
- With the `--params` option, as a JSON object or reference to a JSON file.

For example, review this plan:

```plaintext
plan example::test_params(Targetspec $nodes, String $command){
    run_command($command, $nodes)
}
```
You can pass parameters using either option below:

- puppet plan run example::test_params nodes=my-node.company.com
command=whoami

- puppet plan run example::test_params --params '{"nodes":"my-
node.company.com", "command":"whoami"}'

You can't combine these two ways of passing in parameters. Choose either inline or --params.

If you use the inline way, parameter types other than string, integer, double, and Boolean will be interpreted as strings. Use the --params method if you want them read as their original type.

**Inspecting plans**

View the plans you have installed and which ones you have permissions to run, as well as individual plan metadata.

Log into your primary server or client tools workstation and run one of the following commands to see information about your plan inventory:

<table>
<thead>
<tr>
<th>Command</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet plan show</td>
<td>View a list of your permitted plans.</td>
</tr>
<tr>
<td>puppet plan show --all</td>
<td>View a list of all installed plans.</td>
</tr>
<tr>
<td>puppet plan show &lt;PLAN NAME&gt;</td>
<td>View plan metadata. The output includes the plan's required command format and available parameters.</td>
</tr>
</tbody>
</table>

For example, this plan allows a $nodes parameter and a $version parameter, specified as data types TargetSpec and Integer.

```plaintext
plan infra::upgrade_apache {
    TargetSpec $nodes,
    Integer $version,
    run_task('package', $nodes, name => 'apache', action => 'upgrade', version => $version)
}
```

To view plan metadata in the console, choose which plan you want to run in the Plan field and click on the View plan metadata link.

**Running plans alongside code deployments**

The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

**Before you begin**

Plans in Puppet Enterprise (PE) require Managing code with Code Manager on page 774.

You might want to disable the file sync locking mechanism if:

- You want to allow code deployments to complete while plans are running.
- You want to allow plans to start while code deployments are in progress.
- Your code deployments don't frequently or substantially change the environment state that plans run in.
- You aren't concerned if the environment state changes (due to a concurrent code deployment) during a plan run.

⚠️ **CAUTION:**
When you disable the file sync locking mechanism, the environment states your plans run in might be inconsistent or change while the plans are starting, running, or finishing. This depends on when your code deployments happen and whether they happen while a plan is running. Puppet functions and plans that call other plans might behave unexpectedly if a code deployment occurs while a plan is running.

If it is important to you that your plans always run in a consistent environment state, you probably don't want to disable the file sync locking mechanism.

1. In the PE console, go to Node groups > PE Infrastructure > PE Orchestrator.

2. On the Classes tab, locate (or add) the puppet_enterprise::profile::plan_executor class, and set the versioned_deploys parameter to true. The full declaration is:

   ```
puppet_enterprise::profile::plan_executor::versioned_deploys: true
   ```

   **Important:** Setting this parameter to true disables the file sync client's locking mechanism that usually enforces a consistent environment state for your plans. The locking mechanism prevents plans from starting while a code deployment occurs and forces code deployments to wait while a plan is in progress.

   Tasks, scripts, and apply block compilations always use the latest synced version of your code, regardless of this setting. However, after you set versioned_deploys to true, Puppet functions and plans that call other plans also use the latest synced version of your code, instead of the version of the code that was present when the plan started. Due to the possibility for the code to change during the plan run, Puppet functions and plans that call other plans might behave unexpectedly if a code deployment occurs while a plan is running.

   If you want to enforce a consistent environment state for plans, set versioned_deploys to false.

   If Code deployments time out on page 804 while waiting for long-running plans to finish, adjust the timeouts_sync setting in your Code Manager parameters on page 788.

3. Commit your changes.

4. The orchestrator server doesn't automatically restart after setting this parameter, so you must restart the pe-orchestration-services service to finish applying the change. To do this, run the following command on the primary server:

   ```
service pe-orchestration-services reload
   ```

Plans and code deployments now start and finish without blocking each other.

**Related information**

- **Declare classes** on page 464
- **Lockless code deploys** on page 781

Writing plans

Plans allow you to run more than one task with a single command, compute values for the input to a task, and run other complex workflows at the same time. They also allow you greater flexibility for creating custom RBAC limitations by wrapping tasks and other workflows within plans.
Writing plans in Puppet language
Writing plans in the Puppet language gives you better error handling and more sophisticated control than YAML plans. Plans written in the Puppet language also allow you to apply blocks of Puppet code to remote targets.

Writing plans in YAML
YAML plans run a list of steps in order, which allows you to define simple workflows. Steps can contain embedded Puppet code expressions to add logic where necessary.

Naming plans
Name plans according to the module name, file name, and path to ensure code readability.

Place plan files in your module’s ./plans directory, using these file extensions:

- Puppet plans — .pp
- YAML plans — .yaml, not .yml

Plan names are composed of two or more name segments, indicating:

- The name of the module the plan is located in.
- The name of the plan file, without the extension.
- If the plan is in a subdirectory of ./plans, the path within the module.

For example, given a module called mymodule with a plan defined in ./mymodule/plans/myplan.pp, the plan name is mymodule::myplan.

A plan defined in ./mymodule/plans/service/myplan.pp would be mymodule::service::myplan. Use the plan name to refer to the plan when you run commands.

The plan filename init is special because the plan it defines is referenced using the module name only. For example, in a module called mymodule, the plan defined in init.pp is the mymodule plan.

Avoid giving plans the same names as constructs in the Puppet language. Although plans don’t share their namespace with other language constructs, giving plans these names makes your code difficult to read.

Each plan name segment:

- Must begin with a lowercase letter.
- Can include lowercase letters, digits, or underscores.
- Must not be a reserved word.
- Must not have the same name as any Puppet data types.
- Namespace segments must match the regular expression \A[a-z][a-z0-9_]\Z

Defining plan permissions
RBAC for plans is distinct from RBAC for individual tasks. This distinction means that a user can be excluded from running a certain task, but still have permission to run a plan that contains that task.

The RBAC structure for plans allows you to write plans with more robust, custom control over task permissions. Instead of allowing a user free rein to run a task that can potentially damage your infrastructure, you can wrap a task in a plan and only allow them to run it under circumstances you control.

For example, if you are configuring permissions for a new user to run plan infra::upgrade_git, you can allow them to run the package task but limit it to the git package only.

```c
plan infra::upgrade_git {
    TargetSpec $targets,
    Integer $version,
} {
    run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```
Use parameter types to fine-tune access

Parameter types provide another layer of control over user permissions. In the upgrade_git example above, the plan only provides access to the git package, but the user can choose whatever version of git they want. If there are known vulnerabilities in some versions of the git package, you can use parameter types like Enum to restrict the version parameter to versions that are safe enough for deployment.

For example, the Enum restricts the $version parameter to versions 1:2.17.0-1ubuntu1 and 1:2.17.1-1ubuntu0.4 only.

```puppet
plan infra::upgrade_git (
    TargetSpec $targets,
    Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
    run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```

You can also use PuppetDB queries to select parameter types.

For example, if you need to restrict the targets that infra::upgrade_git can run on, use a PuppetDB query to identify which targets are selected for the git upgrade.

```puppet
plan infra::upgrade_git (
    Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
    # Use puppetdb to find the nodes from the "other" team's web cluster
    $query = [from, nodes, ['=', [fact, cluster], "other_team"]]
    $selected_nodes = puppetdb_query($query).map() |$target| {
        $target[certname]
    }
    run_task('package', $selected_nodes, name => 'git', action => 'upgrade', version => $version)
}
```

Specifying plan parameters

Specify plan parameters to do things like determine which targets to run different parts of your plan on. You can pass a parameter as a single target name, comma-separated list of target names, Target data type, or array. The target names can be either certnames or inventory node names.

The example plan below shows the target parameters $load_balancers and $webservers specified as data type TargetSpec. The plan then calls the run_task function to specify which targets to run the tasks on. The Target names are collected and stored in $webserver_names by iterating over the list of Target objects returned by get_targets. Task parameters are serialized to JSON format so that extracting the names into an array of strings ensures that the webservers parameter is in a format that can be converted to JSON.

```puppet
plan mymodule::my_plan(
    TargetSpec $load_balancer,
    TargetSpec $webservers,
) {
    # Extract the Target name from $webservers
    $webserver_names = get_targets($webservers).map |$n| { $n.name }
    # process webservers
    run_task('mymodule::lb_remove', $load_balancer, webservers => $webserver_names)
    run_task('mymodule::update_frontend_app', $webservers, version => '1.2.3')
    run_task('mymodule::lb_add', $load_balancer, webservers => $webserver_names)
}
```
To execute this plan from the command line, pass the parameters as parameter=value. The Targetspec accepts either an array as JSON or a comma separated string of target names.

```
puppet plan run mymodule::myplan
load_balancer=lb.myorg.com
webservers='"kermit.myorg.com","gonzo.myorg.com"'
```

Alternatively, here is an example of the same plan, run on the same targets, using the plan run API.

```
curl -k -X POST -H "Content-Type: application/json" \
-H "X-Authentication:$TOKEN" \
-d '{ "environment": "$ENV", \
"plan_name": "mymodule::myplan", \
"params": {"targets": "$TARGET_NAME", \
"load_balancer": "lb.myorg.com", \
"webservers": ["kermit.myorg.com", "gonzo.myorg.com"]} }' \
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"
```

Parameters that are passed to the run_* plan functions are serialized to JSON. For example, in the plan below, the default value of $example_nul is undef. The plan calls the test::demo_undef_bash with the example_nul parameter.

```
plan test::parameter_passing ( 
   TargetSpec $targets, 
   Optional[String[1]] $example_nul = undef, 
) { 
   return run_task('test::demo_undef_bash', $targets, example_nul => $example_nul) 
}
```

The implementation of the demo_undef_bash.sh task is:

```
#!/bin/bash
example_env=$PT_example_nul
echo "Environment: $PT_example_nul"
echo "Stdin:
    cat -
```

By default, the task expects parameters passed as a JSON string on stdin to be accessible in prefixed environment variables.

Additionally, you can use the plan_run API with token authentication.

```
curl -k -X POST -H "Content-Type: application/json" \
-H "X-Authentication:$TOKEN" \
-d '{ "environment": "$ENV", \
"plan_name": "test::parameter_passing", \
"params": {"targets": "$TARGET_NAME"} }' \
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"
```

**Using Hiera data in plans**

Use the lookup() function in plans to look up Hiera data. You can look up data inside or outside of apply blocks, or use the plan_hierarchy key to look up data both inside and outside apply blocks within the same plan.

Inside apply blocks, PE compiles catalogs for each target and has unlimited access to your Hiera data. You can use the same Hiera configuration, data, and lookup process as you do throughout PE.

Outside apply blocks, the plan executes a script, doesn’t have a concept of a target or context, and cannot load per-target data. These limitations make some common Hiera features, like interpolating target facts, incompatible with plans in PE outside of apply blocks.
You can look up static Hiera data outside of apply blocks by adding a `plan_hierarchy` key to your Hiera configuration at the same level as the hierarchy key. This allows you to look up data inside and outside apply blocks in the same plan, enabling you to use your existing Hiera configuration in plans without encountering an error if per-target interpolations exist and your plan tries to look up data outside an apply block.

Static Hiera data is also useful for user-specific data that you want the plan to look up.

For example, consider the Hiera configuration below at `<ENV_DIR>/hiera.yaml`.

```yaml
version: 5
hierarchy:
  - name: "Target specific data"
    path: "targets/%{trusted.certname}.yaml"
  - name: "Per-OS defaults"
    path: "os/%{facts.os.family}.yaml"
  - name: Common
    path: hierarchy.yaml

plan_hierarchy:
  - name: Common
    path: plan_hierarchy.yaml
```

You can set a user-specific API key in the `plan_hierarchy.yaml` data file, as well as use Hiera to look up a per-target filepath inside an apply block by using the following pieces of data:

Use the following data located at `<ENV_DIR>/data/plan_hierarchy.yaml`:

```yaml
api_key: 12345
```

Use this data located at `<ENV_DIR>/data/targets/myhost.com`:

```yaml
confpath: "C:\Program Files\Common Files\mytool.conf"
```

As a result, the plan looks up the API key in the first lookup() call, and the target-specific data inside the apply block:

```ruby
plan plan_lookup(
  TargetSpec $targets
)
{
  $outside_apply = lookup('api_key')
  run_task("make_request", $targets, 'api_key' => $outside_apply)
  $in_apply = apply($targets) {
    file { ${confpath}:
      ensure => file,
      content => "setting: false"
    }
  }
}
```

**Target objects**

The Target object represents a target and its specific connection options.

The state of a target is stored in the code for the duration of a plan, allowing you to collect facts or set variables for a target and retrieve them later. Target objects must reference a target in the PE inventory. This includes targets connected via the PCP protocol that have puppet-agent installed, or targets in the PE inventory added with either SSH or WinRM credentials or as network devices. Target objects in PE do not have control over their connection information, and the connection info cannot be changed from within a plan.

Because target objects in PE are references, and cannot control their own configuration, accessing target connection info will return empty data.
**TargetSpec**

The TargetSpec type is a wrapper for defining targets that allows you to pass a target, or multiple targets, into a plan. Use TargetSpec for plans that accept a set of targets as a parameter to ensure clean interaction with the CLI and other plans.

TargetSpec accepts strings allowed by --targets, a single target object, or an array of targets and target patterns. To operate on an individual target, resolve the target to a list via get_targets.

For example, to loop over each target in a plan, accept a TargetSpec argument, but call get_targets on it before looping.

```plaintext
plan loop(TargetSpec $targets) {
  get_targets($targets).each |$target| {
    run_task('my_task', $target)
  }
}
```

**Set variables and facts on targets**

You can use the $target.facts() and $target.vars() functions to set transport configuration values, variables, and facts from a plan. Facts come from running facter or another fact collection application on the target, or from a fact store like PuppetDB. Variables are computed externally or assigned directly.

For example, set variables in a plan using $target.set_var:

```plaintext
plan vars(String $host) {
  $target = get_targets($host)[0]
  $target.set_var('newly_provisioned', true)
  $targetvars = $target.vars
  run_command("echo 'Vars for $host: ${targetvars}'", $host)
}
```

Or set variables in the inventory file using the vars key at the group level.

```plaintext
groups:
  - name: my_targets
targets:
    - localhost
  vars:
    operatingsystem: windows
    config:
      transport: ssh
```

**Collect facts from targets**

The facts plan connects to targets, discovers facts, and stores these facts on the targets.

The plan uses these methods to collect facts:

- On ssh targets, it runs a Bash script.
- On winrm targets, it runs a PowerShell script.
- On pcp or targets where the Puppet agent is present, it runs Facter.

For example, use the facts plan to collect facts and then uses those facts to decide which task to run on the targets.

```plaintext
plan run_with_facts(TargetSpec $targets) {
  # This collects facts on targets and update the inventory
  run_plan(facts, targets => $targets)
  $centos_targets = get_targets($targets).filter |$n| { $n.facts['os'][['name']] == 'CentOS' }
```
Collect facts from PuppetDB

You can use the `puppetdb_fact` plan to collect facts for targets when they are running a Puppet agent and sending facts to PuppetDB.

For example, use the `puppetdb_fact` plan to collect facts, and then use those facts to decide which task to run on the targets.

```plaintext
plan run_with_facts(TargetSpec $targets) {
  # This collects facts on targets and update the inventory
  run_plan(puppetdb_fact, targets => $targets)

  $centos_targets = get_targets($targets).filter |$n| { $n.facts['os'] ['name'] == 'CentOS' }
  $ubuntu_targets = get_targets($targets).filter |$n| { $n.facts['os'] ['name'] == 'Ubuntu' }
  run_task(centos_task, $centos_targets)
  run_task(ubuntu_task, $ubuntu_targets)
}
```

Collect general data from PuppetDB

You can use the `puppetdb_query` function in plans to make direct queries to PuppetDB.

For example, you can discover targets from PuppetDB and then run tasks on them. You must configure the PuppetDB client before running it. See the PQL tutorial to learn how to structure pql queries and see the PQL reference guide for query examples.

```plaintext
plan pdb_discover {
  $result = puppetdb_query("inventory[certname] { app_role == 'web_server' }")
  # extract the certnames into an array
  $names = $result.map |$r| { $r['certname'] } |
  # wrap in url. You can skip this if the default transport is pcp
  $targets = $names.map |$n| { "pcp://$n" }
  run_task('my_task', $targets)
}
```

Returning results from plans

Use the `return` function to return results that you can use in other plans or save for other uses.

Any plan that does not call the `return` function returns `undef`.

For example,

```plaintext
plan return_result(
  $targets
) {
  return run_task('mytask', $targets)
}
```

The result of a plan must match the `PlanResult` type alias. This includes JSON types as well as the plan language types, which have well defined JSON.

- Undef
• String
• Numeric
• Boolean
• Target
• ApplyResult
• Result
• ResultSet
• Error
• Array with only PlanResult
• Hash with String keys and PlanResult values

or

```
Variant[Data, String, Numeric, Boolean, Error, Result, ResultSet, Target, Array[Boltlib::PlanResult], Hash[String, Boltlib::PlanResult]]
```

### Plan errors and failure
Any plan that completes execution without an error is considered successful. There are some specific scenarios that always cause a plan failure, such as calling the `fail_plan` function.

#### Plan failure due to absent `_catch_errors` option
If you call some functions without the `_catch_errors` option and they fail on any target, the plan itself fails. These functions include:

- `upload_file`
- `run_command`
- `run_script`
- `run_task`
- `run_plan`

If there is a plan failure due to an absent `_catch_errors` option when using `run_plan`, any calling plans also halt until a `run_plan` call with `_catch_errors` or a `catch_errors` block is reached.

### Failing a plan
If you are writing a plan and think it’s failing, you can fail the plan with the `fail_plan` function. This function fails the plan and prevents calling plans from executing any further, unless `run_plan` was called with `_catch_errors` or in a `catch_errors` block.

For example, use the `fail_plan` function to pass an existing error or create a new error with a message that includes the kind, details, or issue code.

```
fail_plan('The plan is failing', 'mymodules/pear-shaped', {'failednodes' => $result.error_set.names})
```

### Catching errors in plans
When you use the `catch_errors` function, it executes a block of code and returns any errors, or returns the result of the block if no errors are raised.

Here is an example of the `catch_errors` function.

```
plan test (String[1] $role) {
    $result_or_error = catch_errors(['pe/puppetdb-error']) || {
        puppetdb_query("inventory[certname] { app_role == ${role} }")
    }
}
```
If there is an error in a plan, it returns the Error data type, which includes:

- **msg**: The error message string.
- **kind**: A string that defines the kind of error similar to an error class.
- **details**: A hash with details about the error from a task or from information about the state of a plan when it fails, for example, `exit_code` or `stack_trace`.
- **issue_code**: A unique code for the message that can be used for translation.

Use the Error data type in a case expression to match against different kinds of errors. To recover from certain errors and fail on others, set up your plan to include conditionals based on errors that occur while your plan runs. For example, you can set up a plan to retry a task when a timeout error occurs, but fail when there is an authentication error.

The first plan below continues whether it succeeds or fails with a `mymodule/not-serious` error. Other errors cause the plan to fail.

```ruby
plan mymodule::handle_errors {
  $result = run_plan('mymodule::myplan', '_catch_errors' => true)
  case $result {
    Error['mymodule/not-serious'] : { notice("$result.message") }
    Error : { fail_plan($result) } } run_plan('mymodule::plan2')
}
```

### Puppet and Ruby functions in plans

You can package some common general logic in plans using Puppet language and Ruby functions; however, some functions are not allowed. You can also call plan functions, such as `run_task` or `run_plan`, from within a function.

These Puppet language constructs are not allowed in plans:

- Defined types.
- Classes.
- Resource expressions, such as `file { title: mode => '0777' }`
- Resource default expressions, such as `File { mode => '0666' }`
- Resource overrides, such as `File['/tmp/foo'] { mode => '0444' }`
- Relationship operators: `-> <- ~> <~`
- Functions that operate on a catalog: `include, require, contain, create_resources`
- Collector expressions, such as `SomeType < | | >, SomeType <= | | >=`
- ERB templates.

Additionally, there are some nuances of the Puppet language to keep in mind when writing plans:

- The `--strict_variables` option is on, so if you reference a variable that is not set, you get an error.
- The `--strict-error` option is on, so minor language issues generate errors. For example `{ a => 10, a => 20 }` is an error because there is a duplicate key in the hash.
- Most Puppet settings are empty and not configurable when using plans in PE.
- Logs include "source location" (file, line) instead of resource type or name.

**Handling plan function results**

Each execution function, or a function you use to operate on one or more targets, returns a ResultSet. Each target you executed on returns a Result. The apply action returns a ResultSet containing ApplyResult objects.

You can iterate on an instance of ResultSet as if it were an Array[Variant[Result, ApplyResult]]. This means iterative functions like each, map, reduce, or filter work directly on the ResultSet returning each result.

A ResultSet may contain these functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>names()</td>
<td>Names all targets in the set as an Array.</td>
</tr>
<tr>
<td>empty()</td>
<td>Returns Boolean if the execution result set is empty.</td>
</tr>
<tr>
<td>count()</td>
<td>Returns an Integer count of targets.</td>
</tr>
<tr>
<td>first()</td>
<td>Specifies the first Result object, useful to unwrap single results.</td>
</tr>
<tr>
<td>find(String $target_name)</td>
<td>Specifies the Result for a specific target.</td>
</tr>
<tr>
<td>error_set()</td>
<td>Returns a ResultSet containing only the results of failed targets.</td>
</tr>
<tr>
<td>ok_set()</td>
<td>Returns a ResultSet containing only the successful results.</td>
</tr>
<tr>
<td>filter_set(block)</td>
<td>Filters a ResultSet with the given block and returns a ResultSet object (where the filter function returns an array or hash).</td>
</tr>
<tr>
<td>targets()</td>
<td>Specifies an array of all the Target objects from every Result in the set.</td>
</tr>
<tr>
<td>ok()</td>
<td>Specifies a Boolean that is the same as error_set.empty.</td>
</tr>
<tr>
<td>to_data()</td>
<td>Returns an array of hashes representing either Result or ApplyResults.</td>
</tr>
</tbody>
</table>

A Result may contain these functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>value()</td>
<td>Specifies the hash containing the value of the Result.</td>
</tr>
<tr>
<td>target()</td>
<td>Specifies the Target object that the Result is from.</td>
</tr>
<tr>
<td>error()</td>
<td>Returns an Error object constructed from the _error in the value.</td>
</tr>
<tr>
<td>message()</td>
<td>Specifies the _output key from the value.</td>
</tr>
<tr>
<td>ok()</td>
<td>Returns true if the Result was successful.</td>
</tr>
<tr>
<td>[]</td>
<td>Accesses the value hash directly.</td>
</tr>
<tr>
<td>to_data()</td>
<td>Returns a hash representation of Result.</td>
</tr>
<tr>
<td>action()</td>
<td>Returns a string representation of result type (task, command, etc.).</td>
</tr>
</tbody>
</table>
An ApplyResult may contain these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>report()</td>
<td>Returns the hash containing the Puppet report from the application.</td>
</tr>
<tr>
<td>target()</td>
<td>Returns the Target object that the Result is from.</td>
</tr>
<tr>
<td>error()</td>
<td>Returns an Error object constructed from the _error in the value.</td>
</tr>
<tr>
<td>ok()</td>
<td>Returns true if the Result was successful.</td>
</tr>
<tr>
<td>to_data()</td>
<td>Returns a hash representation of ApplyResult.</td>
</tr>
<tr>
<td>action()</td>
<td>Returns a string representation of result type (apply).</td>
</tr>
</tbody>
</table>

For example, to check if a task ran correctly on all targets, and the check fails if the task fails:

```erb
$r = run_task('sometask', ..., '_catch_errors' => true)
unless $r.ok {
  fail("Running sometask failed on the targets ${r.error_set.names}")
}
```

You can do iteration and check if the result is an error. This example outputs feedback about the result of a task.

```erb
$r = run_task('sometask', ..., '_catch_errors' => true)
$r.each |$result| {
  $target = $result.target.name
  if $result.ok {
    out::message("$target returned a value: ${result.value}"")
  } else {
    out::message("$target errored with a message: ${result.error.message}"")
  }
}
```

Similarly, you can iterate over the array of hashes returned by calling to_data on a ResultSet and access hash values. For example,

```erb
$r = run_command('whoami')
$r.to_data.each |$result_hash| { notice($result_hash['result']['stdout']) } 
```

You can also use filter_set to filter a ResultSet and apply a ResultSet function such as targets to the output:

```erb
$filtered = $result.filter_set |$r| {
  $r['tag'] == "you're it"
}.targets
```

**Applying manifest blocks from a plan**

You can apply manifest blocks, or chunks of Puppet code, to remote systems during plan execution using the apply and apply_prep functions.

You can create manifest blocks that use existing content from the Forge, or use a plan to mix procedural orchestration and action with declarative resource configuration from a block. Most features of the Puppet language are available in a manifest block.

If your plan includes a manifest block, use the apply_prep function in your plan *before* your manifest block. The apply_prep function syncs and caches plugins and gathers facts by running Facter, making the facts available to the manifest block.
For example:

```plaintext
apply_prep($target)
apply($target) { notify { foo: } }
```

**Note:** You can use apply and apply_prep only on targets connected via PCP.

### apply options

The apply function supports these options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_catch_errors</td>
<td>true</td>
<td>Returns a ResultSet, including failed results, rather than failing the plan. Boolean.</td>
</tr>
<tr>
<td>_description</td>
<td>none</td>
<td>Adds a description to the apply block. String.</td>
</tr>
<tr>
<td>_noop</td>
<td>true</td>
<td>Applies the manifest block in no-operation mode, returning a report of changes it would make but does not take action. Boolean.</td>
</tr>
</tbody>
</table>

For example,

```plaintext
# Preview installing docker as root on $targets.
apply($targets, _catch_errors => true, _noop => true) {
  include 'docker'
}
```

### How manifest blocks are applied

When you apply a manifest code from a plan, the manifest code and any facts generated for each target are sent to Puppet Server for compilation. During code compilation, variables are generated in the following order:

1. Facts gathered from the targets set in your inventory.
2. Local variables from the plan.
3. Variables set in your inventory.

After a successful compilation, PE copies custom module content from the module path and applies the catalog to each target. After the catalog is executed on each target, apply generates and returns a report about each target.

### Return value

The apply function returns a ResultSet object that contains an ApplyResult object for each target.

For example:

```plaintext
镊results = apply($targets) { ... }
镊results.each |$result| {
  out::message($result.report)
}
```

### Using Hiera data in a manifest block

Hiera is a key-value configuration data look up system, used for separating data from Puppet code. Use Hiera data to implicitly override default class parameters. You can also explicitly look up data from Hiera via the lookup function.
Note: Plans in PE currently only support Hiera version 5.

For example:

```ruby
plan do_thing() {
  apply('node1.example.com') {
    notice("Some data in Hiera: \${lookup('mydata')}\")
  }
}
```

**Plan logging**

You can view plan run information in log files or printed to a terminal session using the `out::message` function or built-in Puppet logging functions.

**Outputting to the CLI or console**

Use `out::message` to display output from plans. This function always prints message strings to STDOUT regardless of the log level and doesn't log them to the log file. When using `out::message` in a plan, the messages are visible on the Plan details page in the console.

**Puppet log functions**

In addition to `out::message`, you can use Puppet logging functions. Puppet logs messages to `/var/log/puppetlabs/orchestration-services/orchestration-services.log`

When using Puppet logging, each command's usual logging level is downgraded by one level except for `warn` and `error`.

For example, here are the Puppet logging commands with their actual level when used in plans.

```ruby
...
warning('logging text') - logs at warn level
err('logging text') - logs at error level
notice('logging text') - logs at info level
info('logging text') - logs at debug level
debug('logging text') - logs at trace level
...
```

The log level for `orchestration-services.log` is configured with normal levels. For more information about log levels for Bolt, see Puppet log functions in Bolt.

**Default action logging**

PE logs plan actions through the `upload_file`, `run_command`, `run_script`, or `run_task` functions. By default, it logs an info level message when an action starts and another when it completes. You can pass a description to the function to replace the generic log message.

```ruby
run_task(my_task, $targets, "Better description", param1 => "val")
```

If your plan contains many small actions, you might want to suppress these messages and use explicit calls to the Puppet log functions instead. To do this, wrap actions in a `without_default_logging` block, which logs action messages at info level instead of notice.

For example, you can loop over a series of targets without logging each action.

```ruby
plan deploy( TargetSpec $targets) {
  without_default_logging() || {
    get_targets($targets).each |$target| {
      run_task(deploy, $target)
    }
  }
}
```
To avoid complications with parser ambiguity, always call `without_default_logging()` and empty block args `||`. 

Correct example

```plaintext
without_default_logging() || { run_command('echo hi', $targets) }
```

Incorrect example

```plaintext
without_default_logging { run_command('echo hi', $targets) }
```

### Example plans

Check out some example plans for inspiration when writing your own.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>facts module</td>
<td>Contains tasks and plans to discover facts about target systems.</td>
<td>Getting started</td>
</tr>
<tr>
<td>facts plan</td>
<td>Gathers facts using the facts task and sets the facts in inventory.</td>
<td>Getting started</td>
</tr>
<tr>
<td>facts::info plan</td>
<td>Uses the facts task to discover facts and map relevant fact values to targets.</td>
<td>Getting started</td>
</tr>
<tr>
<td>reboot module</td>
<td>Contains tasks and plans for managing system reboots.</td>
<td>Intermediate</td>
</tr>
<tr>
<td>reboot plan</td>
<td>Restarts a target system and waits for it to become available again.</td>
<td>Intermediate</td>
</tr>
<tr>
<td>Introducing Masterless Puppet with Bolt</td>
<td>Blog post explaining how plans can be used to deploy a load-balanced web server.</td>
<td>Advanced</td>
</tr>
<tr>
<td>profiles::nginx_install plan</td>
<td>Shows an example plan for deploying Nginx.</td>
<td>Advanced</td>
</tr>
</tbody>
</table>

- **Getting started** resources show simple use cases such as running a task and manipulating the results.
- **Intermediate** resources show more advanced features in the plan language.
- **Advanced** resources show more complex use cases such as applying puppet code blocks and using external modules.

### Writing plans in YAML

YAML plans run a list of steps in order, which allows you to define simple workflows. Steps can contain embedded Puppet code expressions to add logic where necessary.

**Note:** YAML plans are an experimental feature and might experience breaking changes in future minor releases.

### Naming plans

Name plans according to the module name, file name, and path to ensure code readability.

Place plan files in your module's `.plans` directory, using these file extensions:

- Puppet plans — `.pp`
- YAML plans — `.yaml`, not `.yml`
Plan names are composed of two or more name segments, indicating:

- The name of the module the plan is located in.
- The name of the plan file, without the extension.
- If the plan is in a subdirectory of ./plans, the path within the module.

For example, given a module called mymodule with a plan defined in ./mymodule/plans/myplan.pp, the plan name is mymodule::myplan.

A plan defined in ./mymodule/plans/service/myplan.pp would be mymodule::service::myplan.

Use the plan name to refer to the plan when you run commands.

The plan filename init is special because the plan it defines is referenced using the module name only. For example, in a module called mymodule, the plan defined in init.pp is the mymodule plan.

Avoid giving plans the same names as constructs in the Puppet language. Although plans don't share their namespace with other language constructs, giving plans these names makes your code difficult to read.

Each plan name segment:

- Must begin with a lowercase letter.
- Can include lowercase letters, digits, or underscores.
- Must not be a reserved word.
- Must not have the same name as any Puppet data types.
- Namespace segments must match the regular expression `\A[a-z][a-z0-9_]*\Z`

**Defining plan permissions**

RBAC for plans is distinct from RBAC for individual tasks. This distinction means that a user can be excluded from running a certain task, but still have permission to run a plan that contains that task.

The RBAC structure for plans allows you to write plans with more robust, custom control over task permissions. Instead of allowing a user free rein to run a task that can potentially damage your infrastructure, you can wrap a task in a plan and only allow them to run it under circumstances you control.

For example, if you are configuring permissions for a new user to run plan infra::upgrade_git, you can allow them to run the package task but limit it to the git package only.

```puppet
plan infra::upgrade_git (
    TargetSpec $targets,
    Integer $version,
) {
    run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```

**Use parameter types to fine-tune access**

Parameter types provide another layer of control over user permissions. In the upgrade_git example above, the plan only provides access to the git package, but the user can choose whatever version of git they want. If there are known vulnerabilities in some versions of the git package, you can use parameter types like Enum to restrict the version parameter to versions that are safe enough for deployment.

For example, the Enum restricts the $version parameter to versions 1:2.17.0-1ubuntu1 and 1:2.17.1-1ubuntu0.4 only.

```puppet
plan infra::upgrade_git (
    TargetSpec $targets,
    Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
    run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```
You can also use PuppetDB queries to select parameter types. For example, if you need to restrict the targets that `infra::upgrade_git` can run on, use a PuppetDB query to identify which targets are selected for the git upgrade.

```yaml
plan infra::upgrade_git {
  Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
}
# Use puppetdb to find the nodes from the "other" team's web cluster
$query = [from, nodes, ['=', [fact, cluster], "other_team"]]
$selected_nodes = puppetdb_query($query).map() |$target| {
  $target[certname]
}
run_task('package', $selected_nodes, name => 'git', action => 'upgrade', version => $version)
```

**Plan structure**

YAML plans contain a list of steps with optional parameters and results.

YAML maps accept these keys:

- **steps**: The list of steps to perform
- **parameters**: (Optional) The parameters accepted by the plan
- **return**: (Optional) The value to return from the plan

**Steps key**

The `steps` key is an array of step objects, each of which corresponds to a specific action to take.

When the plan runs, each step is executed in order. If a step fails, the plan halts execution and raises an error containing the result of the step that failed.

Steps use these fields:

- **name**: A unique name that can be used to refer to the result of the step later
- **description**: (Optional) An explanation of what the step is doing

Other available keys depend on the type of step.

**Command step**

Use a `command` step to run a single command on a list of targets and save the results, containing stdout, stderr, and exit code.

The step fails if the exit code of any command is non-zero.

Command steps use these fields:

- **command**: The command to run
- **target**: A target or list of targets to run the command on

For example:

```yaml
steps:
  - command: hostname -f
    target:
      - web1.example.com
      - web2.example.com
      - web3.example.com
    description: "Get the webserver hostnames"
```

**Task step**

Use a `task` step to run a Bolt task on a list of targets and save the results.
Task steps use these fields:

- **task**: The task to run
- **target**: A target or list of targets to run the task on
- **parameters**: (Optional) A map of parameter values to pass to the task

For example:

```yaml
steps:
  - task: package
target:
    - web1.example.com
    - web2.example.com
    - web3.example.com
description: "Check the version of the openssl package on the webservers"
parameters:
  action: status
  name: openssl
```

**Script step**

Use a script step to run a script on a list of targets and save the results.

The script must be in the files/ directory of a module. The name of the script must be specified as `<modulename>/path/to/script`, omitting the files directory from the path.

Script steps use these fields:

- **script**: The script to run
- **target**: A target or list of targets to run the script on
- **arguments**: (Optional) An array of command-line arguments to pass to the script

For example:

```yaml
steps:
  - script: mymodule/check_server.sh
target:
    - web1.example.com
    - web2.example.com
    - web3.example.com
description: "Run mymodule/files/check_server.sh on the webservers"
arguments:
  - "/index.html"
  - 60
```

**File upload step**

Use a file upload step to upload a file to a specific location on a list of targets.

The file to upload must be in the files/ directory of a Puppet module. The source for the file must be specified as `<modulename>/path/to/file`, omitting the files directory from the path.

File upload steps use these fields:

- **source**: The location of the file to be uploaded
- **destination**: The location to upload the file to

For example:

```yaml
steps:
  - source: mymodule/motd.txt
destination: /etc/motd
Plan step

Use a plan step to run another plan and save its result.

Plan steps use these fields:

- **plan**: The name of the plan to run
- **parameters** (Optional) A map of parameter values to pass to the plan

For example:

```yaml
steps:
  - plan: facts
    description: "Gather facts for the webservers using the built-in facts plan"
    parameters:
      nodes:
        - web1.example.com
        - web2.example.com
        - web3.example.com
```

Resources step

Use a resources step to apply a list of Puppet resources. A resource defines the desired state for part of a target. Bolt ensures each resource is in its desired state. Like the steps in a plan, if any resource in the list fails, the rest are skipped.

For each resources step, Bolt executes the `apply_prep` plan function against the targets specified with the `targets` field. For more information about `apply_prep` see the Applying manifest block section.

Resources steps use these fields:

- **resources**: An array of resources to apply
- **target**: A target or list of targets to apply the resources on

Each resource is a YAML map with a type and title, and optionally a `parameters` key. The resource type and title can either be specified separately with the `type` and `title` keys, or can be specified in a single line by using the type name as a key with the title as its value.

For example:

```yaml
steps:
  - resources:
      # This resource is type 'package' and title 'nginx'
      - package: nginx
        parameters:
          ensure: latest
      # This resource is type 'service' and title 'nginx'
      - type: service
        title: nginx
        parameters:
          ensure: running
      target:
        - web1.example.com
        - web2.example.com
        - web3.example.com
```
Parameters key

Plans accept parameters with the parameters key. The value of parameters is a map, where each key is the name of a parameter and the value is a map describing the parameter.

Parameter values can be referenced from steps as variables.

Parameters use these fields:

- **type**: (Optional) A valid Puppet data type. The value supplied must match the type or the plan fails.
- **default**: (Optional) Used if no value is given for the parameter
- **description**: (Optional)

For example, this plan accepts a `load_balancer` name as a string, two sets of nodes called `frontends` and `backends`, and a `version` string:

```yaml
parameters:
  load_balancer:
    type: String
    description: "The load balancer"
  frontends:
    type: TargetSpec
    description: "The frontend web servers"
  backends:
    type: TargetSpec
    description: "The backend application servers"
  version:
    type: String
    description: "The new application version to deploy"
```

How strings are evaluated

The behavior of strings is defined by how they're written in the plan.

- **single-quoted strings** are treated as string literals without any interpolation.
- **double-quoted strings** are treated as Puppet language double-quoted strings with variable interpolation.
- block-style strings are treated as expressions of arbitrary Puppet code. Note the string itself must be on a new line after the `|` character.
- bare strings are treated dynamically based on their content. If they begin with a $, they're treated as Puppet code expressions. Otherwise, they're treated as YAML literals.

Here's an example of different kinds of strings in use:

```yaml
parameters:
  message:
    type: String
    default: "hello"

steps:
- eval: hello
  description: 'This will evaluate to: hello'
- eval: $message
  description: 'This will evaluate to: hello'
- eval: "$message"
  description: 'This will evaluate to: $message'
- eval: "${message} world"
  description: 'This will evaluate to: hello world'
- eval: 
  $message, $message, $message\].join(" ")
  description: 'This will evaluate to: hello hello hello'
```
Using variables and simple expressions

The simplest way to use a variable is to reference it directly by name. For example, this plan takes a parameter called `nodes` and passes it as the target list to a step:

```yaml
parameters:
  nodes:
    type: TargetSpec

steps:
  - command: hostname -f
    target: $nodes
```

Variables can also be interpolated into string values. The string must be double-quoted to allow interpolation. For example:

```yaml
parameters:
  username:
    type: String

steps:
  - task: echo
    message: "hello ${username}
    target: $nodes
```

Many operations can be performed on variables to compute new values for step parameters or other fields.

Indexing arrays or hashes

You can retrieve a value from an Array or a Hash using the `[]` operator. This operator can also be used when interpolating a value inside a string.

```yaml
parameters:
  users:
    # Array[String] is a Puppet data type representing an array of strings
    type: Array[String]

steps:
  - task: user::add
    target: 'host.example.com'
    parameters:
      name: $users[0]
  - task: echo
    target: 'host.example.com'
    parameters:
      message: "hello ${users[0]}
```

Calling functions

You can call a built-in Bolt function or Puppet function to compute a value.

```yaml
parameters:
  users:
    type: Array[String]

steps:
  - task: user::add
    parameters:
      name: $users.first
  - task: echo
    message: "hello ${users.join(',')}"
```
Using code blocks

Some Puppet functions take a block of code as an argument. For instance, you can filter an array of items based on the result of a block of code.

The result of the `filter` function is an array here, not a string, because the expression isn't inside quotes:

```yaml
parameters:
  numbers:
    type: Array[Integer]

steps:
- task: sum
  description: "add up the numbers > 5"
  parameters:
    indexes: $numbers.filter |$num| ( $num > 5 )
```

Connecting steps

You can connect multiple steps by using the result of one step to compute the parameters for another step.

name key

The `name` key makes its results available to later steps in a variable with that name.

This example uses the `map` function to get the value of `stdout` from each command result and then joins them into a single string separated by commas:

```yaml
parameters:
  nodes:
    type: TargetSpec

steps:
- name: hostnames
  command: hostname -f
  target: $nodes
- task: echo
  parameters:
    message: $hostnames.map |$hostname_result|
    { $hostname_result['stdout'] }.join(',')
```

eval step

The `eval` step evaluates an expression and saves the result in a variable. This is useful to compute a variable to use multiple times later.

```yaml
parameters:
  count:
    type: Integer

steps:
- name: double_count
  eval: $count * 2
- task: echo
  target: web1.example.com
  parameters:
    message: "The count is ${count}, and twice the count is ${double_count}"
Returning results
You can return a result from a plan by setting the `return` key at the top level of the plan. When the plan finishes, the `return` key is evaluated and returned as the result of the plan. If no `return` key is set, the plan returns `undef`.

```yaml
steps:
  - name: hostnames
    command: hostname -f
    target: $nodes

return: $hostnames.map |$hostname_result| { $hostname_result['stdout'] }
```

Applying manifest blocks from a plan
You can apply manifest blocks, or chunks of Puppet code, to remote systems during plan execution using the `apply` and `apply_prep` functions.

You can create manifest blocks that use existing content from the Forge, or use a plan to mix procedural orchestration and action with declarative resource configuration from a block. Most features of the Puppet language are available in a manifest block.

If your plan includes a manifest block, use the `apply_prep` function in your plan before your manifest block. The `apply_prep` function syncs and caches plugins and gathers facts by running Facter, making the facts available to the manifest block.

For example:

```puppet
apply_prep($target)
apply($target) { notify { foo: } }
```

**Note:** You can use `apply` and `apply_prep` only on targets connected via PCP.

**apply options**
The `apply` function supports these options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_catch_errors</code></td>
<td>true</td>
<td>Returns a ResultSet, including failed results, rather than failing the plan. Boolean.</td>
</tr>
<tr>
<td><code>_description</code></td>
<td>none</td>
<td>Adds a description to the apply block. String.</td>
</tr>
<tr>
<td><code>_noop</code></td>
<td>true</td>
<td>Applies the manifest block in no-operation mode, returning a report of changes it would make but does not take action. Boolean.</td>
</tr>
</tbody>
</table>

For example,

```puppet
# Preview installing docker as root on $targets.
apply($targets, _catch_errors => true, _noop => true) {
  include 'docker'
}
```

How manifest blocks are applied
When you apply a manifest code from a plan, the manifest code and any facts generated for each target are sent to Puppet Server for compilation. During code compilation, variables are generated in the following order:
1. Facts gathered from the targets set in your inventory.
2. Local variables from the plan.
3. Variables set in your inventory.

After a successful compilation, PE copies custom module content from the module path and applies the catalog to each target. After the catalog is executed on each target, apply generates and returns a report about each target.

**Return value**

The `apply` function returns a `ResultSet` object that contains an `ApplyResult` object for each target.

For example:

```plaintext
$results = apply($targets) { ... }
$results.each |$result| {
    out::message($result.report)
}
```

**Using Hiera data in a manifest block**

Hiera is a key-value configuration data lookup system, used for separating data from Puppet code. Use Hiera data to implicitly override default class parameters. You can also explicitly look up data from Hiera via the `lookup` function.

**Note:** Plans in PE currently only support Hiera version 5.

For example:

```plaintext
plan do_thing() {
    apply('node1.example.com') {
        notice("Some data in Hiera: ${lookup('mydata')}")
    }
}
```

**Computing complex values**

To compute complex values, you can use a Puppet code expression as the value of any field of a step except the name.

Bolt loads the plan as a YAML data structure. As it executes each step, it evaluates any expressions embedded in the step. Each plan parameter and the values of every previous named step are available in scope.

This lets you take advantage of the power of Puppet language in the places it's necessary, while keeping the rest of your plan simple.

When your plans need more sophisticated control flow or error handling beyond running a list of steps in order, it's time to convert them to Puppet Language plans.

**Converting YAML plans to Puppet language plans**

You can convert a YAML plan to a Puppet language plan with the `bolt plan convert` command.

```plaintext
bolt plan convert path/to/my/plan.yaml
```

This command takes the relative or absolute path to the YAML plan to be converted and prints the converted Puppet language plan to stdout.

**Note:** Converting a YAML plan might result in a Puppet plan which is syntactically correct, but behaves differently. Always manually verify a converted Puppet language plan's functionality. There are some constructs that do not translate from YAML plans to Puppet language plans. These are listed [TODO: insert link to section below!] below. If you convert a YAML plan to Puppet and it changes behavior, file an issue in Bolt's Git repo.
For example, with this YAML plan:

```yaml
# site-modules/mymodule/plans/yamlplan.yaml
parameters:
  nodes:
    type: TargetSpec
steps:
  - name: run_task
    task: sample
    target: $nodes
    parameters:
      message: "hello world"
return: $run_task
```

Run the following conversion:

```bash
$ bolt plan convert site-modules/mymodule/plans/yamlplan.yaml
# WARNING: This is an autogenerated plan. It may not behave as expected.
plan mymodule::yamlplan(
  TargetSpec $nodes
) {
  $run_task = run_task('sample', $nodes, {'message' => "hello world"})
  return $run_task
}
```

**Quirks when converting YAML plans to Puppet language**

There are some quirks and limitations associated with converting a plan expressed in YAML to a plan expressed in the Puppet language. In some cases it is impossible to accurately translate from YAML to Puppet. In others, code that is generated from the conversion is syntactically correct but not idiomatic Puppet code.

**Named `eval` step**

The `eval` step allows snippets of Puppet code to be expressed in YAML plans. When converting a multi-line `eval` step to Puppet code and storing the result in a variable, use the `with lambda`.

For example, here is a YAML plan with a multi-line `eval` step:

```yaml
parameters:
  foo:
    type: Optional[Integer]
    description: foo
    default: 0
steps:
  - eval: |
      $x = $foo + 1
      $x * 2
    name: eval_step
return: $eval_step
```

And here is the same plan, converted to the Puppet language:

```puppet
plan yaml_plans::with_lambda(
  Optional[Integer] $foo = 0
) {
  $eval_step = with() || {
    $x = $foo + 1
    $x * 2
  }
  return $eval_step
}
```
Writing this plan from scratch using the Puppet language, you would probably not use the lambda. In this example the converted Puppet code is correct, but not as natural or readable as it could be.

**Resource step variable interpolation**

When applying Puppet resources in a resource step, variable interpolation behaves differently in YAML plans and Puppet language plans. To illustrate this difference, consider this YAML plan:

```yaml
steps:
  - target: localhost
description: Apply a file resource
resources:
  - type: file
title: '/tmp/foo'
parameters:
  content: $facts['os']['family']
  ensure: present
  - name: file_contents
description: Read contents of file managed with file resource
eval: >
    file::read('/tmp/foo')
return: $file_contents
```

This plan performs apply_prep on a localhost target. Then it uses a Puppet file resource to write the OS family discovered from the Puppet $facts hash to a temporary file. Finally, it reads the value written to the file and returns it. Running bolt plan convert on this plan produces this Puppet code:

```puppet
plan yaml_plans::interpolation_pp() {
  apply_prep('localhost') {
    $interpolation = apply('localhost') {
      file { '/tmp/foo':
        content => $facts['os']['family'],
        ensure => 'present',
      }
    }
    $file_contents = file::read('/tmp/foo')
    return $file_contents
  }
}
```

This Puppet language plan works as expected, whereas the YAML plan it was converted from fails. The failure stems from the $facts variable being resolved as a plan variable, instead of being evaluated as part of compiling the manifest code in an applyblock.

**Dependency order**

The resources in a resources list are applied in order. It is possible to set dependencies explicitly, but when doing so you must refer to them in a particular way. Consider the following YAML plan:

```yaml
parameters:
  nodes:
    type: TargetSpec
steps:
  - name: pkg
target: $nodes
resources:
  - title: openssh-server
type: package
```

© 2022 Puppet, Inc., a Perforce company
Executing this plan fails during catalog compilation because of how Bolt parses the resources referenced in the before and require parameters. You will see the error message

`Could not find resource 'File[/etc/ssh/sshd_config]' in parameter 'before'. The solution is to not quote the resource titles:

```
parameters:
  ensure: present
  before: File['/etc/ssh/sshd_config']
- title: /etc/ssh/sshd_config
type: file
parameters:
  ensure: file
  mode: '0600'
  content: ''
  require: Package['openssh-server']
```

In general, declare resources in order. This is an unusual example to illustrate a case where parameter parsing leads to non-intuitive results.

**Orchestrator API v1 endpoints**

Use this API to gather details about the orchestrator jobs you run.
• **Forming orchestrator API requests** on page 696
  Instructions on interacting with this API.
  • **Command endpoints** on page 697
    Use the /command endpoint to start and stop orchestrator jobs for tasks and plans.
  • **Events endpoints** on page 715
    Use the /events endpoint to learn about events that occurred during an orchestrator job.
  • **Inventory endpoints** on page 721
    Use the /inventory endpoint to discover which nodes can be reached by the orchestrator.
  • **Jobs endpoints** on page 723
    Use the /jobs endpoint to examine orchestrator jobs and their details.
  • **Scheduled jobs endpoints** on page 735
    Use the /scheduled_jobs endpoint to gather information about orchestrator jobs scheduled to run.
  • **Plans endpoints** on page 740
    Use the /plans endpoint to see all known plans in your environments.
  • **Plan jobs endpoints** on page 743
    Use the /plan_jobs endpoint to view details about plan jobs you have run.
  • **Tasks endpoints** on page 750
    Use the /tasks endpoint to view details about the tasks pre-installed by PE and those you've installed.
  • **Root endpoints** on page 754
    Use the /orchestrator endpoint to return metadata about the orchestrator API.
  • **Usage endpoints** on page 755
    Use the /usage endpoint to view details about the active nodes in your deployment.
  • **Scopes endpoints** on page 757
    Use the scopes endpoint to retrieve resources for task constraints.
  • **Orchestrator API error responses** on page 760
    From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**Related information**

**API index** on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Forming orchestrator API requests**

Instructions on interacting with this API.

By default, the orchestrator service listens on port 8143, and all endpoints are relative to the /orchestrator/v1 path. So, for example, the full URL for the jobs endpoint on localhost would be https://localhost:8143/orchestrator/v1/jobs.

**Tip:** The orchestrator API accepts well-formed HTTP(S) requests.

**Authenticating to the orchestrator API with a token**

You need to authenticate requests to the orchestrators’ API. You can do this using user authentication tokens.

For detailed information about authentication tokens, see **Token-based authentication** on page 347. The following example shows how to use a token in an API request.

To use the jobs endpoint of the orchestrator API to get a list of all jobs that exist in the orchestrator, along with their associated metadata, you'd first generate a token with the puppet-access tool. You’d then copy that token and replace <TOKEN> with that string in the following request:

```bash
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print print server):8143/orchestrator/v1/jobs"
```
curl --insecure --header "\$auth_header" "\$uri"

See Usage notes for curl examples for information about forming curl commands.

Example token usage: deploy an environment

If you want to deploy an environment with the orchestrator's API, you can form a request with the token you generated earlier. For example:

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/deploy"
data='{"environment": "production", "scope": { "node_group": "00000000-0000-4000-8000-000000000000" }}'
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

This returns a JSON structure that includes a link to the new job started by the orchestrator.

```
{
  "job": {
    "id": "https://orchestrator.vm:8143/orchestrator/v1/jobs/81",
    "name": "81"
  }
}
```

You can make an additional request to get more information about the job. For example:

```bash
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs/81"
curl --insecure --header "$auth_header" "$uri"
```

Related information

Token-based authentication on page 347

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Command endpoints

Use the /command endpoint to start and stop orchestrator jobs for tasks and plans.

**POST /command/deploy**

Run the orchestrator across all nodes in an environment.

**Request format**

The request body must be a JSON object using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>The environment to deploy. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>scope</td>
<td>Object, required unless target is specified. The PuppetDB query, a list of nodes, or a classifier node group id to deploy.</td>
</tr>
<tr>
<td>description</td>
<td>String, a description of the job.</td>
</tr>
</tbody>
</table>
### Key Definition

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>noop</td>
<td>Boolean, whether to run the agent in no-op mode. The default is false.</td>
</tr>
<tr>
<td>no_noop</td>
<td>Boolean, whether to run the agent in enforcement mode. Defaults to false. This flag overrides noop = true if set in the agent's puppet.conf and cannot be set to true at the same time as the noop flag.</td>
</tr>
<tr>
<td>concurrency</td>
<td>The maximum number of nodes to run at one time. The default is a range between 1 and the value of the global_concurrent_compiles parameter. For information about the global_concurrent_compiles parameter, refer to Configuring orchestration on page 302.</td>
</tr>
<tr>
<td>enforce_environment</td>
<td>Boolean, whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be false if environment is an empty string</td>
</tr>
<tr>
<td>debug</td>
<td>Boolean, whether to use the --debug flag on Puppet agent runs.</td>
</tr>
<tr>
<td>trace</td>
<td>Boolean, whether to use the --trace flag on Puppet agent runs.</td>
</tr>
<tr>
<td>evaltrace</td>
<td>Boolean, whether to use the --evaltrace flag on Puppet agent runs.</td>
</tr>
<tr>
<td>filetimeout</td>
<td>Integer, sets the --filetimeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_connect_timeout</td>
<td>Integer, sets the --http_connect_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_keepalive_timeout</td>
<td>Integer, sets the --http_keepalive_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>http_read_timeout</td>
<td>Integer, sets the --http_read_timeout flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>ordering</td>
<td>String, sets the --ordering flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>skip_tags</td>
<td>String, sets the --skip_tags flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>tags</td>
<td>String, sets the --tags flag on Puppet agent runs to the provided value.</td>
</tr>
<tr>
<td>use_cached_catalog</td>
<td>Boolean, whether to use the --use_cached_catalog flag on Puppet agent runs.</td>
</tr>
<tr>
<td>usecacheonfailure</td>
<td>Boolean, whether to use the --usecacheonfailure flag on Puppet agent runs.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

For example, to deploy the node1.example.com environment in no-op mode, the following request is valid:

```json
{
    "environment" : "",
```
"enforce_environment": false,
"noop": true,
"scope": {
  "nodes": ["node1.example.com"]
},
"userdata": {
  "servicenow_ticket": "INC0011211"
}
}

Scope

Scope is a JSON object containing exactly one of these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>The name of an application or application instance to deploy. If an application type is specified, all instances of that application are deployed.</td>
</tr>
<tr>
<td>nodes</td>
<td>A list of node names to target.</td>
</tr>
<tr>
<td>query</td>
<td>A PuppetDB or PQL query to use to discover nodes. The target is built from certname values collected at the top level of the query.</td>
</tr>
<tr>
<td>node_group</td>
<td>A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group.</td>
</tr>
</tbody>
</table>

To deploy an application instance in the production environment:

```json
{
  "environment": "production",
  "scope": {
    "application": "Wordpress_app[demo]"
  }
}
```

To deploy a list of nodes:

```json
{
  "environment": "production",
  "scope": {
    "nodes": ["node1.example.com", "node2.example.com"]
  }
}
```

To deploy a list of nodes with the certname value matching a regex:

```json
{
  "environment": "production",
  "scope": {
    "query": ["from", "nodes", ["~", "certname", ".*"]]
  }
}
```
To deploy to the nodes defined by the "All Nodes" node group:

```json
{
    "environment" : "production",
    "scope" : {
        "node_group" : "00000000-0000-4000-8000-000000000000"
    }
}
```

**Response format**

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response is a JSON object containing a link to retrieve information about the status of the job and uses any one of these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL that links to the newly created job.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the newly created job.</td>
</tr>
</tbody>
</table>

For example:

```json
{
    "job" : {
        "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234"
    
    "name" : "1234"
    }
}
```

**Error responses**

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the environment does not exist, the server returns a 404 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-environment</td>
<td>If the environment requested contains no applications or no nodes, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-target</td>
<td>If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/dependency-cycle</td>
<td>If the application code contains a cycle, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/puppetdb-error</td>
<td>If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/query-error</td>
<td>If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

**Related information**

[Forming orchestrator API requests](#) on page 696

Instructions on interacting with this API.

[Orchestrator API error responses](#) on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**POST /command/schedule_deploy**

Schedule a Puppet run on a set of nodes.

### Request format

The request body must be a JSON object. The following keys are required:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>The environment to deploy. Required string.</td>
</tr>
<tr>
<td>scope</td>
<td>A PuppetDB query, a list of nodes, or a classifier node group ID to deploy. Required object.</td>
</tr>
<tr>
<td>scheduled_time</td>
<td>Timestamp for when to run the scheduled job. If the timestamp is in the past, a 400 error will be thrown. Required ISO-8601 timestamp.</td>
</tr>
</tbody>
</table>

The following keys are optional:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>concurrency</td>
<td>The maximum number of nodes to run at once. The default, if unspecified, is unlimited. Integer.</td>
</tr>
<tr>
<td>debug</td>
<td>Whether to use the --debug flag on Puppet agent runs. Boolean.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the job. String.</td>
</tr>
<tr>
<td>enforce_environment</td>
<td>Whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be false if environment is an empty string. Boolean.</td>
</tr>
<tr>
<td>evaltrace</td>
<td>Whether to use the --evaltrace flag on Puppet agent runs. Boolean.</td>
</tr>
<tr>
<td>filetimeout</td>
<td>Sets the --filetimeout flag on Puppet agent runs to the provided value. Integer.</td>
</tr>
<tr>
<td>http_connect_timeout</td>
<td>Sets the --http_connect_timeout flag on Puppet agent runs to the provided value. Integer.</td>
</tr>
<tr>
<td>http_keepalive_timeout</td>
<td>Sets the --http_keepalive_timeout flag on Puppet agent runs to the provided value. Integer.</td>
</tr>
<tr>
<td>http_read_timeout</td>
<td>Sets the --http_read_timeout flag on Puppet agent runs to the provided value. Integer.</td>
</tr>
<tr>
<td>noop</td>
<td>Whether to run the agent in no-op mode. Defaults to false. Boolean.</td>
</tr>
<tr>
<td>no_noop</td>
<td>Whether to run the agent in enforcement mode. Defaults to false. This flag overrides noop = true if set in the agent’s configuration and cannot be set to true at the same time as the noop flag. Boolean.</td>
</tr>
<tr>
<td>ordering</td>
<td>Sets the --ordering flag to the provided value on Puppet agent runs. String.</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduled_time</td>
<td>The timestamp for when to run the scheduled job. If the timestamp is in the past, a 400 error will be thrown. Required ISO-8601 timestamp.</td>
</tr>
<tr>
<td>schedule_options</td>
<td>Object.</td>
</tr>
<tr>
<td>interval</td>
<td>Object.</td>
</tr>
<tr>
<td>units</td>
<td>Enum in seconds.</td>
</tr>
<tr>
<td>value</td>
<td>Positive integer.</td>
</tr>
<tr>
<td>skip_tags</td>
<td>Sets the <code>--skip_tags</code> flag to the provided value on Puppet agent runs. String.</td>
</tr>
<tr>
<td>tags</td>
<td>Sets the <code>--tags</code> flag to the provided value on Puppet agent runs. String.</td>
</tr>
<tr>
<td>trace</td>
<td>Whether to use the <code>--trace</code> flag on Puppet agent runs. Boolean.</td>
</tr>
<tr>
<td>use_cached_catalog</td>
<td>Whether to use the <code>--use_cached_catalog</code> flag on Puppet agent runs. Boolean.</td>
</tr>
<tr>
<td>usecacheonfailure</td>
<td>Whether to use the <code>--usecacheonfailure</code> flag on Puppet agent runs. Boolean.</td>
</tr>
</tbody>
</table>

To deploy `node1.example.com` in noop mode, the following request is valid:

```json
{
   "environment" : "",
   "enforce_environment": false,
   "noop" : true,
   "scope" : {
      "nodes" : ["node1.example.com"]
   },
   "scheduled_time": "2027-05-05T19:50:08Z",
   "schedule_options": {
      "interval": {
         "units": "seconds",
         "value": 86400
      }
   }
}
```

**Response format**

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response will be a JSON object containing a link to retrieve information about the status of the job. The keys of this object are as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL that links to the newly created job.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the newly created job.</td>
</tr>
</tbody>
</table>

For example:

```json
{
}
```
"job" : { 
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
    "name" : "1234"
  }
}

Error responses

See the error response documentation for the general format of error responses. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Error</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the environment does not exist, the server returns a 404 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-environment</td>
<td>If the environment requested contains no applications or no nodes, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/empty-target</td>
<td>If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/dependency-cycle</td>
<td>If the application code contains a cycle, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/puppetdb-error</td>
<td>If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/query-error</td>
<td>If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

POST /command/stop

Stop an orchestrator job that is currently in progress. Puppet agent runs that are in progress finish, but no new agent runs start. While agents are finishing, the server continues to produce events for the job.

The job transitions to status stopped when all pending agent runs have finished.

This command is idempotent: it can be issued against the same job any number of times.

Request format

The JSON body of the request must contain the ID of the job to stop. The job ID is the same value as the name property returned with the deploy command.

- job: the name of the job to stop.

For example:

```
{  
  "job": "1234"
}
```

Response format

If the job is stopped successfully, the server returns a 202 response. The response uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL that links to the newly created job.</td>
</tr>
</tbody>
</table>
### Key Definition

| **name** | The name of the newly created job. |
| **nodes** | A hash that shows all of the possible node statuses, and how many nodes are currently in that status. |

**For example:**

```json
{
   "job": {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
      "name": "1234",
      "nodes": {
         "new": 5,
         "running": 8,
         "failed": 3,
         "errored": 1,
         "skipped": 2,
         "finished": 5
      }
   }
}
```

### Error responses

For this endpoint, the **kind** key of the error displays the conflict.

<table>
<thead>
<tr>
<th><strong>Key</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If a job name is not valid, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If a job name is unknown, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

### Related information

**Orchestrator API error responses** on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**POST /command/task**

Run a permitted task job across a set of nodes. Any nodes specified in the scope that you do not have permission to run tasks on are excluded.

### Request format

The request body must be a JSON object and uses the following keys:

<table>
<thead>
<tr>
<th><strong>Key</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>environment</strong></td>
<td>The environment to load the task from. The default is production.</td>
</tr>
<tr>
<td><strong>scope</strong></td>
<td>The PuppetDB query, list of nodes, or a node group ID. Application scopes are not allowed for task jobs. This key is required.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>A description of the job.</td>
</tr>
</tbody>
</table>
For example, to run the package task on node1.example.com, the following request is valid:

```json
{
    "environment" : "test-env-1",
    "task" : "package",
    "params" : {
        "action" : "install",
        "name" : "httpd"
    },
    "scope" : {
        "nodes" : ["node1.example.com"]
    }
}
```

**Scope**

Scope is a JSON object containing exactly one of the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>The name of an application or application instance to deploy. If an application type is specified, all instances of that application will be deployed. This key is only supported for the deploy command.</td>
</tr>
<tr>
<td>nodes</td>
<td>An array of node names to target.</td>
</tr>
<tr>
<td>query</td>
<td>A PuppetDB or PQL query to use to discover nodes. The target is built from the certname values collected at the top level of the query.</td>
</tr>
<tr>
<td>node_group</td>
<td>A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group. Any nodes specified in the scope that the user does not have permissions to run the task on are excluded.</td>
</tr>
</tbody>
</table>

To deploy an application instance in the production environment, the following request is valid:

```json
{
    "environment" : "production",
    "scope" : {
        "application" : "Wordpress_app[demo]"
    }
}
```
To run a task on a list of nodes, the following request is valid:

```json
{
  "environment" : "production",
  "scope" : {
    "nodes" : ["node1.example.com", "node2.example.com"]
  }
}
```

To run a task on a list of nodes with the `certname` value matching a regex, the following request is valid:

```json
{
  "environment" : "production",
  "scope" : {
    "query" : ["from", "nodes", ["~", "certname", ".*"]]
  }
}
```

To deploy to the nodes defined by the "All Nodes" node group the following request is valid:

```json
{
  "environment" : "production",
  "scope" : {
    "node_group" : "00000000-0000-4000-8000-000000000000"
  }
}
```

**Targets**

Targets is an array of JSON objects, each containing the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostnames</td>
<td>Array of hostnames that share the same target attributes. These should each match an entry in the node list scope for the job. String. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>password</td>
<td>Password matching the user specified. One of two credential optional, along with <code>private-key-content</code>. String.</td>
</tr>
<tr>
<td>private-key-content</td>
<td>Content of the ssh key used to ssh to the remote node to run on. String.</td>
</tr>
<tr>
<td>port</td>
<td>Specifies which port on the remote node to use to connect. Integer.</td>
</tr>
<tr>
<td>user</td>
<td>User on the remote system to log in as to run the task. String. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>transport</td>
<td>Specify ssh or winrm. String. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>run-as</td>
<td>An optional user to run commands when using ssh. String.</td>
</tr>
<tr>
<td>sudo-password</td>
<td>A password to use when changing users via <code>run-as</code>. String.</td>
</tr>
<tr>
<td>run-as-command</td>
<td>Command to elevate permissions via <code>run-as</code>. String.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>connect-timeout</td>
<td>How long, in seconds, Bolt should wait when establishing connections. Integer.</td>
</tr>
<tr>
<td>tty</td>
<td>Determines if Bolt uses pseudo tty to meet sudoer restrictions. Boolean.</td>
</tr>
<tr>
<td>tmpdir</td>
<td>The directory to upload and execute temporary files on the target. String.</td>
</tr>
<tr>
<td>extensions</td>
<td>List of file extensions that are accepted for scripts or tasks. String.</td>
</tr>
</tbody>
</table>

For example,

```json
[
  { 
    "hostnames": ["sshnode1.example.com", "sshnode2.example.com"],
    "password": "p@ssw0rd",
    "port": 4444,
    "user": "foo",
    "transport": "ssh"
  },
  { 
    "hostnames": ["winrmnode.example.com"],
    "password": "p@ssw0rd!",
    "port": 4444,
    "user": "foo",
    "transport": "winrm"
  }
]
```

**Response format**

If the task is starts successfully, the response will have a 202 status.

The response will be a JSON object containing a link to retrieve information about the status of the job. The keys of this object are:

- **id**: an absolute URL that links to the newly created job.
- **name**: the name of the newly created job. For example:

```json
{
  "job": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
    "name": "1234"
  }
}
```

**Error responses**

For this endpoint, the **kind** key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the environment does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>
### Key | Definition
--- | ---
puppetlabs.orchestrator/empty-target | If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.
puppetlabs.orchestrator/puppetdb-error | If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.
puppetlabs.orchestrator/query-error | If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.
puppetlabs.orchestrator/not-permitted | This error occurs when a user does not have permission to run the task on the requested nodes. Server returns a 403 response.

### Related information

**Orchestrator API error responses** on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**POST /command/task_target**
Create a new task-target, which is a collection of tasks, nodes and node groups that define a permission group. When a user has permissions to run via a given task-target, they are granted permissions to run the given set of tasks, on the set of nodes described by the task-target.

After creating a task-target, use the **POST /roles** on page 368 endpoint to create the role that controls the permissions to execute the task-target. For the complete task-target workflow, see the blog post Puppet Enterprise RBAC API, or how to manage access to tasks.

### Request format

The request body must be a JSON object and uses the following keys:

### Key | Definition
--- | ---
display_name | A required string that is used to name the task-target. No uniqueness requirements

**tasks** | An optional array of tasks that correspond to the task-target. There is no requirement that the tasks correspond to existing tasks. If omitted or empty, all_tasks must be provided and set to true.

**all_tasks** | An optional boolean to determine if all tasks can be run on designated targets. Defaults to false. If omitted, tasks are required to be specified. If both all_tasks is true and valid tasks are supplied, tasks are ignored and all_tasks is set.

**nodes** | An array of certnames that can correspond to both agent and agentless nodes. The user can run the task against the specified nodes. There is no requirement that the nodes correspond to currently available nodes. The array can be empty.

**node_groups** | An array of node-group-ids that describe the set of nodes the task can be run against. The possible nodes described by the node group will be combined with any nodes in the `nodes` array. The array can be empty.

**pql_query** | An optional string that is a single PQL query to fetch nodes that the task-target corresponds to. The query results must contain the certnames key in the results to identify the permitted nodes.
For example:

```json
{
  "display_name": "foo",
  "tasks": ["abc", "def"],
  "nodes": ["node1", "node2"],
  "node_groups": ["00000000-0000-4000-8000-000000000000"]
}

{
  "display_name": "foo",
  "all_tasks": "true",
  "nodes": ["node1", "node2"],
  "node_groups": ["00000000-0000-4000-8000-000000000000"]
}
```

**Response format**

If the task-target is successfully created, the server will return a 200 response with a JSON object containing a link to retrieve information about the task-target. The keys of this object are:

- **id**: an absolute URL to retrieve the individual task-target.
- **name**: a unique identifier for the task-target.

For example:

```json
{
  "task_target": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/1",
    "name": "1"
  }
}
```

**Error responses**

For this endpoint, the **kind** key of the error displays the conflict. See the [Orchestrator API error responses](#) on page 760 documentation for the general format of error responses.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If there is no display name supplied, there are no tasks specified when supplied, task names are not strings, all_tasks is not a boolean when supplied, all_tasks is not set to true when tasks are not supplied, node names are not strings, node-group ids are not strings, or pql_query is not a string the service returns a 400 response.</td>
</tr>
</tbody>
</table>

**Related information**

**POST /roles** on page 368

Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

**POST /command/schedule_task**

Schedule a task to run at a future date and time.

**Request format**

The request body must be a JSON object and uses the following keys:
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>The task to run on the targets. Required.</td>
</tr>
<tr>
<td>params</td>
<td>The parameters to pass to the task. Required.</td>
</tr>
<tr>
<td>scope</td>
<td>The PuppetDB query, list of nodes, or a node group ID. Application scopes are not allowed for task jobs. Required.</td>
</tr>
<tr>
<td>scheduled_time</td>
<td>The ISO-8601 timestamp the determines when to run the scheduled job. If timestamp is in the past, a 400 error is thrown. Required.</td>
</tr>
<tr>
<td>schedule_options</td>
<td>Object.</td>
</tr>
<tr>
<td>interval</td>
<td>Object.</td>
</tr>
<tr>
<td>units</td>
<td>Enum in seconds.</td>
</tr>
<tr>
<td>value</td>
<td>Positive integer.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the job.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment to load the task from. The default is production.</td>
</tr>
<tr>
<td>noop</td>
<td>Whether to run the job in no-op mode. The default is false.</td>
</tr>
</tbody>
</table>

For example, to run the package task on node1.example.com, the following request is valid:

```json
{
    "environment": "test-env-1",
    "task": "package",
    "params": {
        "action": "install",
        "package": "httpd"
    },
    "scope": {
        "nodes": [
            "node1.example.com"
        ]
    },
    "scheduled_time": "2027-05-05T19:50:08Z",
    "schedule_options": {
        "interval": {
            "units": "seconds",
            "value": 86400
        }
    }
}
```

**Response format**

If the task schedules successfully the server returns 202.

The response is a JSON object containing a link to retrieve information about the status of the job. The following keys are available.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL that links to the newly created job.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>The name of the newly created job.</td>
</tr>
</tbody>
</table>

For example:

```
{
  "scheduled_job": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/2",
    "name": "1234"
  }
}
```

**Error responses**

For this endpoint, the `kind` key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/invalid-time</td>
<td>If the <code>scheduled_time</code> provided is in the past, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

**Related information**

[Orchestrator API error responses](#) on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**POST /command/schedule_plan**

Schedule a plan to run at a later time. The body consists of the same fields as `/command/plan_run` and a `scheduled_time` field.

**Request format**

The request body must be a JSON object. The following keys are available.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan</td>
<td>A required string that indicates which plan to run.</td>
</tr>
<tr>
<td>params</td>
<td>A required object that indicates which parameters to pass to the plan.</td>
</tr>
<tr>
<td>scheduled_time</td>
<td>A required ISO 8601 timestamp to indicate when to run the scheduled job.</td>
</tr>
<tr>
<td></td>
<td>If the timestamp is in the past, a 400 error will be thrown.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the job.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment to load the plan from. The default is production.</td>
</tr>
</tbody>
</table>

To run the package task on `node1.example.com` node, the following request is valid.

```
{
  "environment": "test-env-1",
  "plan": "facts",
  "params": {
    "targets": "node1.example.com"
  },
```

© 2022 Puppet, Inc., a Perforce company
"scheduled_time": "2027-05-05T19:50:08Z"
}

**Response format**

If the plan schedules successfully, the server returns 202.

The response is a JSON object containing a link to retrieve information about the status of the job. The keys are as follows.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL that links to the newly created job.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the newly created job.</td>
</tr>
</tbody>
</table>

For example:

{
    "scheduled_job" : {
        "id" : "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/2",
        "name" : "1234"
    }
}

**Error responses**

See Orchestrator API error responses on page 760 for the error response format. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/invalid-time</td>
<td>If the scheduled_time provided is in the past, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

**POST /command/plan_run**

Run a plan via the plan executor.

**Request format**

The request body must be a JSON object. The following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_name</td>
<td>String, the name of the plan to run.</td>
</tr>
<tr>
<td>params</td>
<td>Hash, the parameters the job will use.</td>
</tr>
<tr>
<td>environment</td>
<td>String, environment to load the plan from. The default is production.</td>
</tr>
<tr>
<td>description</td>
<td>String, description of the job.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

To start the canary plan, the following request is valid:

```
{
    "plan_name" : "canary",
    "description" : "Start the canary plan on node1 and node2",
    "params" : {
```
"nodes" : ["node1.example.com", "node2.example.com"],
    "command" : "whoami",
    "canary" : 1
}]

Response format
If the plan starts successfully, status 202 is returned.
The response is a JSON object containing the generated plan job name.
For example:

{
    "name" : "1234"
}

Error responses
See the error response documentation for the general format of error response. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the plan name is not valid the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/not-permitted</td>
<td>This error occurs if a user makes a request they lack permissions for. The server returns a 403 response.</td>
</tr>
</tbody>
</table>

POST /command/environment_plan_run
Run a plan in a specific environment.

Request format
The request body must be a JSON object. The following keys are required:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_name</td>
<td>The name of the plan to run. String.</td>
</tr>
<tr>
<td>params</td>
<td>The parameters the job will use. Parameters can include queries and node groups by passing the type key as part of the hash for a given parameter. Hash.</td>
</tr>
</tbody>
</table>

The following keys are optional:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>The environment to load the plan from. The default is production. String.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the job. String.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

To run example_plan, the following request is valid:

```json
{
    "plan_name" : "example_plan",
}
"description" : "Output 'message' on the targets contained in 'targets' and 'more targets'.",
"params": {
  "message": {
    "value": "hello"
  },
  "example_hash_param": {
    "value": {
      "value": "foo"
    }
  },
  "targets": {
    "type": "query",
    "value": "nodes[certname] { }"
  },
  "more_targets": {
    "type": "node_group",
    "value": "<uuid>"
  }
}
}

In example_plan, parameters adopt the format {<param_name>:{"value": <param_value>}}. Orchestrator passes <param_value> as the parameter's value to the plan. The orchestrator also passes hello as the message parameter's value to the plan. For hash parameters, like example_hash_param, the hash {"value": "foo"} is passed to the plan as the value for example_hash_param. The type key is an optional way to give the orchestrator additional information about the parameter. It is not to be confused with the parameter's type in the plan metadata.

The following are the values for type:

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>The value key is interpreted as a PuppetDB query. Orchestrator executes the query and passes the resulting list of nodes as the parameter's value to the plan. For example, for the targets parameter, if nodes[certname] { } resolves to the list [&quot;foo_node&quot;, &quot;bar_node&quot;], orchestrator passes [&quot;foo_node&quot;, &quot;bar_node&quot;] into the plan as the parameter's value.</td>
</tr>
<tr>
<td>node_group</td>
<td>The value key is interpreted as a node group UUID. Orchestrator fetches all the nodes that are in this node group and passes them as the parameter's value to the plan. For example, for the more_targets parameter, if &lt;uuid&gt; resolves to [&quot;group_node&quot;, &quot;group_node_two&quot;], orchestrator passes that array into the plan as the parameter's value.</td>
</tr>
</tbody>
</table>

Response format

If the plan starts successfully, the response has a 202 status. The response is a JSON object containing the generated plan job name.

For example:

```json
{
  "name": "1234"
}
```
Error response
See the error response documentation for the general format of error response. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the plan name is not valid the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/not-permitted</td>
<td>This error occurs if a user makes a request they lack permissions for. The server returns a 403 response.</td>
</tr>
</tbody>
</table>

Events endpoints
Use the /events endpoint to learn about events that occurred during an orchestrator job.

GET /jobs/:job-id/events
Retrieve all of the events that occurred during a given job.

Parameters
The request accepts this query parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Start the list of events with the nth event.</td>
</tr>
</tbody>
</table>

For example:

https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?start=1272

Response format
The response is a JSON object that details the events in a job, and uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>next-events</td>
<td>A link to the next event in the job.</td>
</tr>
<tr>
<td>items</td>
<td>A list of all events related to the job.</td>
</tr>
<tr>
<td>id</td>
<td>The job ID.</td>
</tr>
<tr>
<td>type</td>
<td>The current status of the event.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the event was created.</td>
</tr>
<tr>
<td>details</td>
<td>Information about the event.</td>
</tr>
<tr>
<td>message</td>
<td>A message about the given event.</td>
</tr>
</tbody>
</table>

For example:

```json
{
  "next-events" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?start=1272"
  },
  "items" : [ {
    "id" : "1267",
    "type" : "node_running",
    "timestamp" : "2016-05-05T19:50:08Z",
```
Event types

The response format for each event contains one of these event types, which is determined by the status of the event.

<table>
<thead>
<tr>
<th>Event type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_errored</td>
<td>Created when there was an error running Puppet on a node.</td>
</tr>
<tr>
<td>node_failed</td>
<td>Created when Puppet failed to run on a node.</td>
</tr>
<tr>
<td>node_finished</td>
<td>Created when puppet ran successfully on a node.</td>
</tr>
<tr>
<td>node_running</td>
<td>Created when Puppet starts running on a node.</td>
</tr>
<tr>
<td>node_skipped</td>
<td>Created when a Puppet run is skipped on a node (for example, if a dependency fails).</td>
</tr>
<tr>
<td>job_aborted</td>
<td>Created when a job is aborted without completing.</td>
</tr>
<tr>
<td>job_stopping</td>
<td>Created when a stop request is received and the job is running.</td>
</tr>
<tr>
<td>job_finished</td>
<td>Created when a job is no longer running. The details contain final state. This should always be the last event for a job.</td>
</tr>
</tbody>
</table>

Error responses

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the start parameter or the job-id in the request are not integers, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

Related information

Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /plan_jobs/job-id/events
Retrieve all of the events that occurred during a given plan job.

Parameters

The request accepts the following query parameter:
Parameter | Definition
--- | ---
start | The lowest database ID of an event that should be shown.

For example:
https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/events?start=1272

**Response format**
The response is a JSON object that derails the events in a plan job. The following keys and their subkeys are used:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next-events</td>
<td>A link to the next event in the job.</td>
</tr>
<tr>
<td>id</td>
<td>The url of the next event.</td>
</tr>
<tr>
<td>event</td>
<td>The next event id.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>A list of all events related to the job.</td>
</tr>
<tr>
<td>id</td>
<td>The id of the event.</td>
</tr>
<tr>
<td>type</td>
<td>The type of event. See event-types.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the event was created.</td>
</tr>
<tr>
<td>details</td>
<td>Details of the event, differs based on the type of the event. See the potential values section for more detailed information.</td>
</tr>
</tbody>
</table>

For example:

```json
{
    "next-events" : {
        "id" : "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/events?start=1272",
        "event" : "1272"
    },
    "items" : [ {
        "id" : "1267",
        "type" : "task_start",
        "timestamp" : "2016-05-05T19:50:08Z",
        "details" : {
            "job-id" : "8765"
        }
    },
    {
        "id" : "1268",
        "type" : "plan_finished",
        "timestamp" : "2016-05-05T19:50:14Z",
        "details" : {
            "plan-id" : "1234",
            "result" : {
                "Plan output"
            }
        }
    }]
}
```
Event types

The response format for each event will contain one of the following event types, which is determined by the status of the event.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_start</td>
<td>Created when a task run is started.</td>
</tr>
<tr>
<td>script_start</td>
<td>Created when a script run as part of a plan is started.</td>
</tr>
<tr>
<td>command_start</td>
<td>Created when a command run as part of a plan is started.</td>
</tr>
<tr>
<td>upload_start</td>
<td>Created when a file upload as part of a plan is started.</td>
</tr>
<tr>
<td>wait_start</td>
<td>Created when a <code>wait_until_available()</code> call as part of a plan is started.</td>
</tr>
<tr>
<td>out_message</td>
<td>Created when <code>out::message</code> is called as part of a plan. <em>details</em> will contain a <code>message</code> key with a value containing the message sent with <code>out::message</code> truncated to 1,024 bytes. The full message content can be obtained using the GET <code>/plan_jobs/job-id/event/event-id</code> on page 720 endpoint.</td>
</tr>
<tr>
<td>apply_start</td>
<td>Created when a puppet apply run as part of a plan is started.</td>
</tr>
<tr>
<td>apply_prep_start</td>
<td>Created when an <code>apply_prep</code> is run as part of a plan is started.</td>
</tr>
<tr>
<td>plan_finished</td>
<td>Created when a plan job successfully finishes.</td>
</tr>
<tr>
<td>plan_failed</td>
<td>Created when a plan job fails.</td>
</tr>
</tbody>
</table>

"Sub" plans

If a plan contains the `run_plan()` function it will begin execution of another "sub" plan during the execution of the plan job. "Sub" plans will not receive their own plan job, they will execute as part of the original plan job. There are specific events for "sub" plans that indicate a plan within a plan has started or finished:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan_start</td>
<td>Created when a new plan was kicked off from the current plan job using the <code>run_plan()</code> function.</td>
</tr>
<tr>
<td>plan_end</td>
<td>Created when a plan started using <code>run_plan()</code> has finished.</td>
</tr>
</tbody>
</table>

**Note:** `plan_end` indicates the end of a "sub" plan (i.e. the plan job has not finished yet, but the "sub" plan has), while `plan_finished` indicates the end of a plan job.

Potential values in *details*

**task and plan actions**
Includes the job-id if the event is a plan action or task (i.e. `task_start`, `command_start`, `apply_start`, etc).

```json
{
    "type": "task_start",
    "details": {
        "job-id": 69
    },
    "id": "80"
}
```

**plan_finished** or **plan_failed**

If the event type is `plan_finished` or `plan_failed`, details will include `plan-id` and `result`.

```json
{
    "type": "plan_finished",
    "details": {
        "result": "plan result",
        "plan-id": "9"
    },
    "id": "81"
}
```

**out_message**

If the event type is `out_message`, details will include `message`.

```json
{
    "type": "out_message",
    "details": {
        "message": "message sent from a plan"
    }
}
```

**plan_start** or **plan_end**

If the event type is `plan_start` or `plan_end`, details will include the plan that ran (plan) and `plan_end` will include `duration`.

**plan_start**:

```json
{
    "type": "plan_start",
    "details": {
        "plan": "test::sub_plan"
    },
    "id": "76"
}
```

**plan_end**:

```json
{
    "type": "plan_end",
    "details": {
        "plan": "test::sub_plan",
        "duration": 0.647551
    }
}
```
Error responses

See the error response documentation for the general format of error responses. For the endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the start parameter or the job-id in the request are not integers, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the plan job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

GET /plan_jobs/:job-id/event/:event-id

Fetch a specific event for a job.

Response format

The response is a JSON object of the event. The following keys are available.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>ID of this event.</td>
</tr>
<tr>
<td>type</td>
<td>Event type.</td>
</tr>
<tr>
<td>timestamp</td>
<td>When the event occurred.</td>
</tr>
<tr>
<td>details</td>
<td>Hash of detail fields specific to the event type.</td>
</tr>
</tbody>
</table>

Note: Unlike the /plan_jobs/:job-id/events endpoint, no truncation of fields occurs.

For example:

```
{
  "id": "1265",
  "type": "out_message",
  "timestamp": "2016-05-05T19:50:06Z",
  "details": {
    "message": "this is an output message"
  }
}
```

Error responses

See the error response documentation for the general format of error responses. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the job-id in the request is not an integer, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the plan job does not exist, the server returns a 404 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/mismatched-job-event-id</td>
<td>If the event-id specified does not belong to the job-id specified, the server returns a 404 response.</td>
</tr>
</tbody>
</table>
Inventory endpoints

Use the /inventory endpoint to discover which nodes can be reached by the orchestrator.

GET /inventory
List all nodes that are connected to the PCP broker.

Response format

The response is a JSON object containing a list of records for connected nodes, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the connected node.</td>
</tr>
<tr>
<td>connected</td>
<td>The connection status is either true or false.</td>
</tr>
<tr>
<td>broker</td>
<td>The PCP broker the node is connected to.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the connection was made.</td>
</tr>
</tbody>
</table>

For example:

```json
{
    "items": [
        {
            "name": "foo.example.com",
            "connected": true,
            "broker": "pcp://broker1.example.com/server",
            "timestamp": "2016-01-02 13:36:41.449Z"
        },
        {
            "name": "bar.example.com",
            "connected": true,
            "broker": "pcp://broker2.example.com/server",
            "timestamp": "2016-01-02 13:39:16.377Z"
        }
    ]
}
```

Error responses

For this endpoint, the kind key of the error displays the conflict.

The server returns a 500 response if the PCP broker can't be reached.

Related information

Orchestrator API error responses on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /inventory/:node
Return information about whether the requested node is connected to the PCP broker.

Response format

The response is a JSON object indicating whether the queried node is connected, using these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the connected node.</td>
</tr>
<tr>
<td>connected</td>
<td>The connection status is either true or false.</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
For example:

```
{
  "name" : "foo.example.com",
  "connected" : true,
  "broker" : "pcp://broker.example.com/server",
  "timestamp" : "2017-03-29T21:48:09.633Z"
}
```
Error responses
For this endpoint, the kind key of the error displays the conflict.
The server returns a 500 response if the PCP broker can't be reached.

Related information
Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Jobs endpoints
Use the /jobs endpoint to examine orchestrator jobs and their details.

GET /jobs
List all of the jobs known to the orchestrator.

Parameters
The request accepts the following query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Return only the most recent n number of jobs.</td>
</tr>
<tr>
<td>offset</td>
<td>Return results offset n records into the result set.</td>
</tr>
<tr>
<td>order_by</td>
<td>Return results ordered by a column. Ordering is available on owner, timestamp, environment, name, and state. Orderings requested on owner are applied to the login subfield of owner.</td>
</tr>
<tr>
<td>order</td>
<td>Indicates whether results are returned in ascending or descending order. Valid values are &quot;asc&quot; and &quot;desc&quot;. Defaults to &quot;asc&quot;</td>
</tr>
<tr>
<td>type</td>
<td>Indicates the type of job. Matches the command that created the job, which is either deploy, task, or plan_task</td>
</tr>
<tr>
<td>task</td>
<td>An optional task name to match against. Partial matches are supported. If type is specified, the task option is only allowed for task and plan_task types.</td>
</tr>
</tbody>
</table>
Orchestrating Puppet runs, tasks, and plans

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_finish_timestamp</td>
<td>Returns only the jobs given at or after a given UTC timestamp.</td>
</tr>
<tr>
<td>max_finish_timestamp</td>
<td>Returns only the jobs given at or before a given UTC timestamp.</td>
</tr>
</tbody>
</table>

For example:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/jobs?
limit=20&offset=20
```

**Response format**

The response is a JSON object that lists orchestrator jobs and associated details and pagination information.

For information about jobs, the following keys are used:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Contains an array of all the known jobs.</td>
</tr>
<tr>
<td>id</td>
<td>An absolute URL to the given job.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the given job.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the job: new, ready, running, stopping, stopped, finished, or failed.</td>
</tr>
<tr>
<td>command</td>
<td>The command that created that job.</td>
</tr>
<tr>
<td>type</td>
<td>The type of job. Either deploy, task, plan_task, plan_script, plan_upload, plan_command, plan_wait, plan_apply, plan_apply_prep</td>
</tr>
<tr>
<td>options</td>
<td>All of the options used to create that job. The schema of options might vary based on the command.</td>
</tr>
<tr>
<td>owner</td>
<td>The subject id and login for the user that requested the job.</td>
</tr>
<tr>
<td>description</td>
<td>(deprecated) A user-provided description of the job. For future compatibility, use the description in options.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the job state last changed.</td>
</tr>
<tr>
<td>created_timestamp</td>
<td>The time the job was created.</td>
</tr>
<tr>
<td>finished_timestamp</td>
<td>The time the job finished.</td>
</tr>
<tr>
<td>duration</td>
<td>The number of seconds the job took to run.</td>
</tr>
<tr>
<td>environment</td>
<td>(deprecated) The environment that the job operates in.</td>
</tr>
<tr>
<td>node_count</td>
<td>The number of nodes the job runs on.</td>
</tr>
<tr>
<td>node_states</td>
<td>A JSON map containing the counts of nodes involved with the job by current node state. Unrepresented states are not displayed. This field is null when no nodes exist for a job.</td>
</tr>
<tr>
<td>nodes</td>
<td>A link to get more information about the nodes participating in a given job.</td>
</tr>
</tbody>
</table>
### Key Definitions

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>report</td>
<td>A link to the report for a given job.</td>
</tr>
<tr>
<td>events</td>
<td>A link to the events for a given job.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

For information about pagination, the following keys are used:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagination</td>
<td>Contains information about the limit, offset, and total number of items.</td>
</tr>
<tr>
<td>limit</td>
<td>The number of items the request was limited to.</td>
</tr>
<tr>
<td>offset</td>
<td>The offset from the start of the collection. Offset begins at zero.</td>
</tr>
<tr>
<td>order_by</td>
<td>Either owner, timestamp, environment, name, or state.</td>
</tr>
<tr>
<td>order</td>
<td>Either asc or desc.</td>
</tr>
<tr>
<td>total</td>
<td>The total number of items in the collection, ignoring limit and offset.</td>
</tr>
<tr>
<td>type</td>
<td>The type of job filtered on, if any. Either deploy, task, or plan_task.</td>
</tr>
</tbody>
</table>

For example:

```json
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
      "name": "1234",
      "state": "finished",
      "command": "deploy",
      "type": "deploy",
      "node_count": 5,
      "node_states": {
        "finished": 2,
        "errored": 1,
        "failed": 1,
        "running": 1
      },
      "options": {
        "concurrency": null,
        "noop": false,
        "trace": false,
        "debug": false,
        "scope": {},
        "enforce_environment": true,
        "environment": "production",
        "evaltrace": false,
        "target": null,
        "description": "deploy the web app"
      },
      "owner": {
        "email": "admin@example.com",
        "is_revoked": false,
      }
    }
  ]
}
```
"last_login": "2020-05-05T14:03:06.226Z",
"is_remote": true,
"login": "admin",
"inherited_role_ids": [2],
"group_ids": ["9a588fd8-3daa-4fc2-a396-bf88945def1e"],
"is_superuser": false,
"id": "751a8f7e-b53a-4cc0-deb6aa0388ec",
"role_ids": [1],
"display_name": "Admin",
"is_group": false,
"description": "deploy the web app",
"timestamp": "2016-05-20T16:45:31Z",
"started_timestamp": "2016-05-20T16:41:15Z",
"finished_timestamp": "2016-05-20T16:45:31Z",
"duration": "256.0",
"environment": {
  "name": "production"
},
"report": {
  "id": "https://localhost:8143/orchestrator/v1/jobs/375/report"
},
"events": {
  "id": "https://localhost:8143/orchestrator/v1/jobs/375/events"
},
"nodes": {
  "id": "https://localhost:8143/orchestrator/v1/jobs/375/nodes"
},
"userdata": {
  "servicenow_ticket": "INC0011211"
}
},
{
  "description": "",
  "report": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1235/report"
  },
  "name": "1235",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1235/events"
  },
  "command": "plan_task",
  "type": "plan_task",
  "state": "finished",
  "nodes": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1235/nodes"
  },
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1235",
  "environment": {
    "name": ""
  },
  "options": {
    "description": "",
    "plan_job": 197,
    "noop": null,
    "task": "facts",
    "sensitive": [],
    "scheduled-job-id": null,
    "params": {},
    "scope": {
Error responses

For this endpoint, the **kind** key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the limit parameter is not an integer, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

Related information

Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**GET /jobs/:job-id**
List all details of a given job.

**Response format**
The response is a JSON object that lists all details of a given job, and uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to the given job.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the given job.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the job: new, ready, running, stopping, stopped, finished, or failed.</td>
</tr>
<tr>
<td>command</td>
<td>The command that created that job.</td>
</tr>
<tr>
<td>type</td>
<td>The type of job. Either deploy, task, plan_task, plan_script, plan_upload, plan_command, plan_wait, plan_apply, or plan_apply_prep</td>
</tr>
<tr>
<td>options</td>
<td>All of the options used to create that job.</td>
</tr>
<tr>
<td>owner</td>
<td>The subject id and login for the user that requested the job.</td>
</tr>
<tr>
<td>description</td>
<td>(deprecated) A user-provided description of the job.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the job state last changed.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment that the job operates in.</td>
</tr>
<tr>
<td>node_count</td>
<td>The number of nodes the job runs on.</td>
</tr>
<tr>
<td>nodes</td>
<td>A link to get more information about the nodes participating in a given job.</td>
</tr>
<tr>
<td>report</td>
<td>A link to the report for a given job.</td>
</tr>
<tr>
<td>events</td>
<td>A link to the events for a given job.</td>
</tr>
<tr>
<td>status</td>
<td>The various enter and exit times for a given job state.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

For example:

```json
{
  "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
  "name" : "1234",
  "command" : "deploy",
  "type" : "deploy",
  "state" : "finished",
  "options" : {
    "concurrency" : null,
    "noop" : false,
    "trace" : false,
    "debug" : false,
    "scope" : {
      "application" : "Wordpress_app"
    },
    "enforce_environment" : true,
    "environment" : "production",
  }
}```
"evaltrace" : false,
"target" : null
},

"node_count" : 5,
"owner" : {
  "email" : "admin@example.com",
  "is_revoked" : false,
  "last_login" : "2020-05-05T14:03:06.226Z",
  "is_remote" : true,
  "login" : "admin",
  "inherited_role_ids" : [ 2 ],
  "group_ids" : [ "9a588f7e-3daa-4fc2-a396-bf88945def1e" ],
  "is_superuser" : false,
  "id" : "751a8f7e-b53a-4cc0-9e93-db6aa38ec",
  "role_ids" : [ 1 ],
  "display_name" : "Admin",
  "is_group" : false
},

"description" : "deploy the web app",
"timestamp": "2016-05-20T16:45:31Z",
"started_timestamp": "2016-05-20T16:41:15Z",
"finished_timestamp": "2016-05-20T16:45:31Z",
"duration": "256.0",
"environment" : {
  "name" : "production"
},

"status" : [ {
  "state" : "new",
  "enter_time" : "2016-04-11T18:44:31Z",
  "exit_time" : "2016-04-11T18:44:31Z"
},
  { "state" : "ready",
  "enter_time" : "2016-04-11T18:44:31Z",
  "exit_time" : "2016-04-11T18:44:31Z"
},
  { "state" : "running",
  "enter_time" : "2016-04-11T18:44:31Z",
  "exit_time" : "2016-04-11T18:45:31Z"
},
  { "state" : "finished",
  "enter_time" : "2016-04-11T18:45:31Z",
  "exit_time" : null
} ],

"nodes" : { "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234/nodes" },

"userdata" : {} }

---

**Error responses**

For this endpoint, the `kind` key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the job-id in the request is not an integer, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>
Related information
Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /jobs/:job-id/nodes
List all of the nodes associated with a given job.

Request format
The request accepts the following optional query parameters:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Returns only the most recent ( n ) number of node results in a job.</td>
</tr>
<tr>
<td>offset</td>
<td>Returns results offset ( n ) records into the result set.</td>
</tr>
<tr>
<td>order_by</td>
<td>Returns results ordered by a column. Ordering is available on name, duration, state, start_timestamp, and finish_timestamp.</td>
</tr>
<tr>
<td>order</td>
<td>Indicates whether results should be returned in ascending (asc) or descending (desc) order. Defaults to asc.</td>
</tr>
<tr>
<td>state</td>
<td>Returns only the nodes in the state supplied. One of either new, ready, running, stopping, stopped, finished, or failed.</td>
</tr>
</tbody>
</table>

Response format
The response is a JSON object that details the nodes associated with a job.

next-events is an object containing information about where to find events about the job. It has the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The url of the next event.</td>
</tr>
<tr>
<td>event</td>
<td>The next event id.</td>
</tr>
</tbody>
</table>

items: a list of all the nodes associated with the job, where each node has the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>(deprecated) The time of the most recent activity on the node. Prefer start_timestamp and finish_timestamp.</td>
</tr>
<tr>
<td>start_timestamp</td>
<td>The time the node starting running or nil if the node hasn't started running or was skipped.</td>
</tr>
<tr>
<td>finish_timestamp</td>
<td>The time the node finished running or nil if the node hasn't finished running or was skipped.</td>
</tr>
<tr>
<td>duration</td>
<td>The duration of the puppet run in seconds if the node has finished running, the duration in seconds that has passed after the node started running if it is currently running, or nil if the node hasn't started running or was skipped</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the job.</td>
</tr>
<tr>
<td>transaction_uuid</td>
<td>(deprecated) The ID used to identify the node's last report.</td>
</tr>
<tr>
<td>name</td>
<td>The hostname of the node.</td>
</tr>
<tr>
<td>details</td>
<td>Information about the last event and state of a given node.</td>
</tr>
<tr>
<td>result</td>
<td>The result of a successful, failed, errored node-run. The schema of this varies.</td>
</tr>
</tbody>
</table>

For example:

```json
{
    "next-events": {
        "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/3/events?start=10",
        "event": "10"
    },
    "items" : [ {
        "timestamp" : "2015-07-13T20:37:01Z",
        "finish_timestamp" : "2015-07-13T20:37:01Z",
        "duration" : 48.0,
        "state" : "state",
        "transaction_uuid" : :uuid,
        "name" : "wss6c3w9wngpycg.example.com",
        "details" : {
            "message": "Message of latest event"
        },
        "result": {
            "output_1": "success",
            "output_2": [1, 1, 2, 3]}
    } ]
}
```

**Results**

The result field is available after a node finishes, fails, or errors and contains the contents of the details of the corresponding event. In task jobs this is the result of executing the task. For puppet jobs it contains metrics from the puppet run.

For example:

An error when running a task:

```json
"result" : {
    "msg" : "Running tasks is not supported for agents older than version 5.1.0",
    "kind" : "puppetlabs.orchestrator/execution-failure",
    "details" : {
        "node" : "copper-6"
    }
}
```
Raw stdout from a task:

```
"result": {
  "output": "test\n"
}
```

Structured output from a task:

```
"result": {
  "status": "up to date",
  "version": "5.0.0.201.g879fc5a-1.el7"
}
```

Error output from a task:

```
"result": {
  "error": "Invalid task name 'package::status'"
}
```

Puppet run results:

```
"result": {
  "hash": "d7ec44e176bb4b2e8a816157ebbae23b065b68cc",
  "noop": {
    "noop": false,
    "no_noop": false
  },
  "status": "unchanged",
  "metrics": {
    "corrective_change": 0,
    "out_of_sync": 0,
    "restarted": 0,
    "skipped": 0,
    "total": 347,
    "changed": 0,
    "scheduled": 0,
    "failed_to_restart": 0,
    "failed": 0
  },
  "environment": "production",
  "configuration_version": "1502024081"
}
```

Using puppet apply as part of a plan results:

```
"result": {
  "noop": false,
  "status": "unchanged",
  "metrics": {
    "corrective_change": 0,
    "out_of_sync": 0,
    "restarted": 0,
    "skipped": 0,
    "total": 347,
    "changed": 0,
    "scheduled": 0,
    "failed_to_restart": 0,
    "failed": 0
  }
}
```
Details

The details field contains information based on the last event and current state of the node and can be empty. In some cases it can duplicate data from the results key for historical reasons.

If the node state is **finished** or **failed** the details hash can include a message and a report-url. (deprecated) for jobs started with the run command it also duplicates some information from the result.

```
{
  "items" : [ {
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "finished",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : { 
      "report-url" : "https://peconsole.example.com/#/cm/report/a15bf509dd7c40705e4e1c24d0935e2e8a1591df",
      "message": "Finished puppet run on wss6c3w9wngpycg.example.com - Success!"
    },
    "result" : { 
      "metrics" : { 
        "total" : 82,
        "failed" : 0,
        "changed" : 51,
        "skipped" : 0,
        "restarted" : 2,
        "scheduled" : 0,
        "out_of_sync" : 51,
        "failed_to_restart" : 0
      },
    },
  } ],
}
```

If the node state is **skipped** or **errored**, the service includes a :detail key that explains why a node is in that state.

```
{
  "items" : [ { 
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "failed",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : { 
      "message": "Error running puppet on wss6c3w9wngpycg.example.com: java.net.Exception: Something went wrong"
    },
  } ],
}
```

If the node state is **running**, the service returns the run-time (in seconds).

```
{
  "items" : [ { 
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "running",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : { 
      ...
    } ]
}
```
"run-time": 30,
   "message": "Started puppet run on wss6c3w9wngpycg.example.com..."
}
},
...
}
}

Error responses
For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the job-id in the request is not an integer, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

Related information
Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /jobs/:job-id/report
Returns a report for a given job.

Response format
The response is a JSON object that reports the status of a job, and uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>An array of all the reports associated with a given job.</td>
</tr>
<tr>
<td>node</td>
<td>The hostname of a node.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the job.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the job was created.</td>
</tr>
<tr>
<td>events</td>
<td>Any events associated with that node during the job.</td>
</tr>
</tbody>
</table>

For example:

```
{
   "items" : [ 
     { 
       "node" : "wss6c3w9wngpycg.example.com",
       "state" : "running",
       "timestamp" : "2015-07-13T20:37:01Z",
       "events" : [ ]
     },
     { 
       "node" : "wss6c3w9wngpycg.example.com",
       "state" : "running",
       "timestamp" : "2015-07-13T20:37:01Z",
       "events" : [ ]
     },
     ...
   ]
}
```

Error responses
For this endpoint, the kind key of the error displays the conflict.
## Related information

**Orchestrator API error responses** on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

### Scheduled jobs endpoints

Use the `/scheduled_jobs` endpoint to gather information about orchestrator jobs scheduled to run.

#### GET /scheduled_jobs

List scheduled jobs in ascending order.

**Parameters**

The request accepts the following query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Return only the most recent ( n ) number of jobs.</td>
</tr>
<tr>
<td>offset</td>
<td>Return results offset ( n ) records into the result set.</td>
</tr>
<tr>
<td>order_by</td>
<td>One of <code>next_run</code> (default), <code>scheduled_time</code>, <code>name</code>.</td>
</tr>
<tr>
<td>order</td>
<td><code>asc</code> (default) or <code>desc</code></td>
</tr>
<tr>
<td>type</td>
<td>The type of job. Matches the command that created the job, one of: <code>deploy</code>, <code>task</code>, or <code>plan</code>.</td>
</tr>
</tbody>
</table>

For example:

```plaintext
https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs?
limit=20&offset=20
```

#### Response format

The response is a JSON object that contains a list of the known jobs and information about the pagination.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Contains an array of all the scheduled jobs.</td>
</tr>
<tr>
<td>id</td>
<td>An absolute URL to the given job.</td>
</tr>
<tr>
<td>name</td>
<td>The ID of the scheduled job</td>
</tr>
<tr>
<td>type</td>
<td>The type of scheduled job (currently only <code>task</code>)</td>
</tr>
<tr>
<td>task</td>
<td>The name of the task associated with the scheduled task job</td>
</tr>
<tr>
<td>plan</td>
<td>The name of the plan associated with the scheduled plan job</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>scope</strong></td>
<td>The specification of the targets for the task.</td>
</tr>
<tr>
<td><strong>environment</strong></td>
<td>The environment that the job operates in.</td>
</tr>
<tr>
<td><strong>owner</strong></td>
<td>The specification for the user that requested the job.</td>
</tr>
<tr>
<td><strong>email</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>login</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>display_name</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>is_revoked</strong></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>last_login</strong></td>
<td>ISO-8601 timestamp</td>
</tr>
<tr>
<td><strong>is_remote</strong></td>
<td>A boolean for the type of user</td>
</tr>
<tr>
<td><strong>is_group</strong></td>
<td>A boolean for the type of user</td>
</tr>
<tr>
<td><strong>is_superuser</strong></td>
<td>A boolean for the type of user</td>
</tr>
<tr>
<td><strong>role_ids</strong></td>
<td>Array</td>
</tr>
<tr>
<td><strong>inherited_role_ids</strong></td>
<td>Array</td>
</tr>
<tr>
<td><strong>group_ids</strong></td>
<td>Array</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>A user-provided description of the job.</td>
</tr>
<tr>
<td><strong>next_run</strong></td>
<td>An ISO-8601 timestamp for the next run of a scheduled job.</td>
</tr>
<tr>
<td><strong>scheduled_time</strong></td>
<td>An ISO8601 timestamp for when the scheduled job runs.</td>
</tr>
<tr>
<td><strong>schedule_options</strong></td>
<td>Object</td>
</tr>
<tr>
<td><strong>interval</strong></td>
<td>Object</td>
</tr>
<tr>
<td><strong>units</strong></td>
<td>Enum [seconds]</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>Positive integer</td>
</tr>
<tr>
<td><strong>noop</strong></td>
<td>Boolean. Is true if the job runs in no-operation mode, false otherwise.</td>
</tr>
<tr>
<td><strong>job_options</strong></td>
<td>Object for the options supplied for the job.</td>
</tr>
<tr>
<td><strong>pagination</strong></td>
<td>Contains the information about the limit, offset and total number of items.</td>
</tr>
<tr>
<td><strong>limit</strong></td>
<td>A restricted number of items for the request to return.</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>A number offset from the start of the collection (zero based).</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td>The total number of items in the collection, ignoring limit and offset.</td>
</tr>
<tr>
<td><strong>order_by</strong></td>
<td>The sort field, one of next_run (default), scheduled_time, or name.</td>
</tr>
<tr>
<td><strong>order</strong></td>
<td>asc or desc</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>The type of job filtered on, if any. One of deploy, task, or plan.</td>
</tr>
</tbody>
</table>
For example:

```json
{
   "items": [
      {
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/1",
         "name": "1",
         "type": "task",
         "task": "echo",
         "scope": {
            "nodes": [
               "east.example.com",
               "west.example.com"
            ],
         },
         "environment": "production",
         "owner": {
            "email": "fred@example.com",
            "is_revoked": false,
            "last_login": "2020-05-08T15:57:28.444Z",
            "is_remote": true,
            "login": "fred",
            "inherited_role_ids": [2],
            "group_ids": [
               "9a588fd8-3daa-4fc2-a396-bf88945def1e"
            ],
            "is_superuser": false,
            "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
            "role_ids": [1],
            "display_name": "Fred",
            "is_group": false
         },
         "description": "rear face the cranfitouser",
         "next_run": "2018-10-12T19:50:08Z",
         "scheduled_time": "2018-10-05T19:50:08Z",
         "schedule_options": {
            "interval": {
               "units": "seconds",
               "value": 604800
            }
         },
         "noop": false,
         "job_options": {
            "noop": false,
            "task": "echo"
         }
      },
      {
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/2",
         "name": "2",
         "type": "deploy",
         "scope": {
            "nodes": ["east.example.com", "west.example.com"
         ]
      }
   ]
}
```

© 2022 Puppet, Inc., a Perforce company
"environment": "production",
"owner": {
    "email": "fred@example.com",
    "is_revoked": false,
    "last_login": "2020-05-08T15:57:28.444Z",
    "is_remote": true,
    "login": "fred",
    "inherited_role_ids": [
        2
    ],
    "group_ids": [
        "9a588fd8-3daa-4fc2-a396-bf88945def1e"
    ],
    "is_superuser": false,
    "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
    "role_ids": [
        1
    ],
    "display_name": "Fred",
    "is_group": false
},
"description": "middle face the cranfitouser",
"next_run": "2019-05-05T19:50:08Z",
"scheduled_time": "2019-05-05T19:50:08Z",
"schedule_options": {},
"noop": false,
"job_options": {
    "noop": false,
    "debug": false,
    "trace": false,
    "no_noop": false,
    "evaltrace": false,
    "concurrency": null,
    "enforce_environment": false
}
},
{
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/3",
    "name": "3",
    "type": "task",
    "task": "facter_task",
    "scope": {
        "query": "inventory[certname] { facts.aio_agent_version ~ "\\d+ \" }"
    },
    "environment": "production",
    "owner": {
        "email": "fred@example.com",
        "is_revoked": false,
        "last_login": "2020-05-08T15:57:28.444Z",
        "is_remote": true,
        "login": "fred",
        "inherited_role_ids": [
            2
        ],
        "group_ids": [
            "9a588fd8-3daa-4fc2-a396-bf88945def1e"
        ],
        "is_superuser": false,
        "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
        "role_ids": [
            1
        ]
    }
}
Error responses

See Orchestrator API error responses on page 760 for the format of error responses. For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the limit or offset parameter is not an integer, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

Related information

Orchestrator API error responses on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

DELETE /scheduled_jobs/:job-id

Delete a scheduled job.

- Response 204
- Response 403
  - Body

```json
{
  "kind": "puppetlabs.orchestrator/not-permitted",
  "msg": "Not authorized to delete job {id}"
}
```

Plans endpoints

Use the /plans endpoint to see all known plans in your environments.

GET /plans

List all known plans in a given environment.

Parameters

The request accepts the following query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>Return the plans in a particular environment. Defaults to production.</td>
</tr>
</tbody>
</table>

Response format

The response is a JSON object that lists the known plans and where to find more information about them. It uses the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Contains an array of all known plans for the specified environment.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environment</td>
<td>A map containing the name key with the environment name and a code_id key indicating the code id the plans are listed from.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> code_id is always null.</td>
</tr>
</tbody>
</table>

Each item above has the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to retrieve plan details.</td>
</tr>
<tr>
<td>name</td>
<td>The full name of the plan.</td>
</tr>
<tr>
<td>permitted</td>
<td>A boolean indicating if the user making the request is permitted to use the plan.</td>
</tr>
</tbody>
</table>

For example:

```json
{
   "environment": {
      "name": "production",
      "code_id": null
   },
   "items": [
      {
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/profile/firewall",
         "name": "profile::firewall",
         "permitted": true
      },
      {
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/profile/rolling_update",
         "name": "profile::rolling_update",
         "permitted": true
      },
      {
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/canary/random",
         "name": "canary::random",
         "permitted": false
      }
   ]
}
```

**Error responses**

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the environment parameter is not a legal environment name, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the specified environment doesn't exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>
GET `/plans/:module/:plannname`
Return data about the specified plan, including metadata.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>Return the plan from a particular environment. Defaults to production.</td>
</tr>
<tr>
<td><strong>Note:</strong> code_id is always null.</td>
<td></td>
</tr>
</tbody>
</table>

**Response format**

The response is a JSON object that includes information about the specified plan. The following keys are used:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to retrieve plan details.</td>
</tr>
<tr>
<td>name</td>
<td>The full name of the plan.</td>
</tr>
<tr>
<td>environment</td>
<td>A map containing a <code>name</code> key with the environment name and a <code>code_id</code> key indicating the code id the plan is being listed from.</td>
</tr>
<tr>
<td>permitted</td>
<td>A boolean indicating if the user is permitted to use the plan or not.</td>
</tr>
<tr>
<td>metadata</td>
<td>A map containing a <code>description</code> field with the plan's description, as well as a <code>parameters</code> map where keys are parameter names and values that map to <code>type</code>, <code>default_value</code>, and <code>description</code>.</td>
</tr>
<tr>
<td>type</td>
<td>The type of the parameter, matches a puppet type. If no type is present for the parameter, defaults to <code>{{Any}}</code>.</td>
</tr>
<tr>
<td>default_value</td>
<td>The default value of the parameter when it is not set. Optional.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the parameter. Optional.</td>
</tr>
</tbody>
</table>

For example:

```
{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/package/install",
  "name": "canary::random",
  "environment": {
    "name": "production",
    "code_id": null
  },
  "metadata": {},
  "permitted": true
}
```

**Error responses**
Plan jobs endpoints

Use the /plan_jobs endpoint to view details about plan jobs you have run.

GET /plan_jobs
List the known plan jobs sorted by name and in descending order.

Parameters

The request accepts the following query parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Return only the most recent $n$ number of jobs.</td>
</tr>
<tr>
<td>offset</td>
<td>Return results offset $n$ records into the result set.</td>
</tr>
<tr>
<td>order_by</td>
<td>Return results ordered by a column. One of either owner, timestamp,</td>
</tr>
<tr>
<td></td>
<td>environment, name, or state. Orderings requesting owner are applied to the</td>
</tr>
<tr>
<td></td>
<td>login subfield of owner.</td>
</tr>
<tr>
<td>order</td>
<td>Indicates whether results are returned in ascending or descending order.</td>
</tr>
<tr>
<td>results</td>
<td>Whether to include or exclude the plan output for each plan in the list.</td>
</tr>
<tr>
<td></td>
<td>The default is include.</td>
</tr>
<tr>
<td>min_finish_timestamp</td>
<td>Returns only the jobs given at or after a given UTC timestamp.</td>
</tr>
<tr>
<td>max_finish_timestamp</td>
<td>Returns only the jobs given at or before a given UTC timestamp.</td>
</tr>
</tbody>
</table>

Response format

The response is a JSON object that contains a list of the known plan jobs, and information about the pagination.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>An array of all the plan jobs.</td>
</tr>
<tr>
<td>id</td>
<td>An absolute URL to the given plan job.</td>
</tr>
<tr>
<td>name</td>
<td>The ID of the plan job.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the plan job: running, success, or failure</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>options</td>
<td>Information about the plan job: description, plan_name, parameters, scheduled_job_id, and environment.</td>
</tr>
<tr>
<td>description</td>
<td>The user-provided description for the plan job.</td>
</tr>
<tr>
<td>plan_name</td>
<td>The name of the plan that was run, for example package::install.</td>
</tr>
<tr>
<td>parameters</td>
<td>The parameters passed to the plan for the job.</td>
</tr>
<tr>
<td>scheduled_job_id</td>
<td>The scheduled_job_id if the plan was run as a scheduled job.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment the plan was run in.</td>
</tr>
<tr>
<td>result</td>
<td>The output from the plan job.</td>
</tr>
<tr>
<td>owner</td>
<td>The subject ID and login for the user that requested the job.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the plan job last changed.</td>
</tr>
<tr>
<td>created_timestamp</td>
<td>The time the plan job was created.</td>
</tr>
<tr>
<td>finished_timestamp</td>
<td>The time the plan job finished.</td>
</tr>
<tr>
<td>duration</td>
<td>The number of seconds the plan job has been running, or total duration of the plan run.</td>
</tr>
<tr>
<td>events</td>
<td>A link to the events for a given plan job.</td>
</tr>
<tr>
<td>status</td>
<td>A hash of jobs that ran as part of the plan job, with associated lists of states and their enter and exit times.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
<tr>
<td>pagination</td>
<td>Contains the information about the limit, offset and total number of items.</td>
</tr>
<tr>
<td>limit</td>
<td>The number of items the request was limited to.</td>
</tr>
<tr>
<td>offset</td>
<td>The offset from the start of the collection (zero based).</td>
</tr>
<tr>
<td>total</td>
<td>The total number of items in the collection, ignoring limit and offset.</td>
</tr>
</tbody>
</table>

For example:

```json
{
  "items": [
    {
      "finished_timestamp": "2020-09-23T18:00:13Z",
      "name": "38",
      "events": {
        "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/38/events"
      },
      "state": "success",
      "result": [
        "orchestrator.example.com: CentOS 7.2.1511 (RedHat)"
      ],
      "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/38",
      "created_timestamp": "2020-09-23T18:00:08Z",
      "duration": "0:00:05.135681"
    }
  ]
}```
"duration": 123.456,
"options": {
  "description": "just the facts",
  "plan_name": "facts::info",
  "parameters": {
    "targets": "orchestrator.example.com"
  },
  "sensitive": [],
  "scheduled_job_id": "116",
  "project": {
    "project_id": "myproject_id",
    "ref": "524df30f58002d30a3549c52c341ccee29da2981"
  }
},
"owner": {
  "email": "",
  "is_revoked": false,
  "last_login": "2020-08-05T17:54:07.045Z",
  "is_remote": false,
  "login": "admin",
  "is_superuser": true,
  "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
  "role_ids": [
    1
  ],
  "display_name": "Administrator",
  "is_group": false
},
"userdata": {
  "servicenow_ticket": "INC0011211"
},
"finished_timestamp": null,
"name": "37",
"events": {
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/37/events"
},
"state": "running",
"id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/37",
"created_timestamp": "2018-06-06T20:22:08Z",
"duration": 123.456,
"options": {
  "description": "Testing myplan",
  "plan_name": "myplan",
  "parameters": {
    "nodes": [
      "orchestrator.example.com"
    ],
    "sensitive": ["secret"],
    "environment": "production",
    "scheduled_job_id": "5"
  }
},
"owner": {
  "email": "",
  "is_revoked": false,
  "last_login": "2018-06-06T20:22:06.327Z",
  "is_remote": false,
  "login": "admin",
  "is_superuser": true,
  "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",

"role_ids": [
  1
],
"display_name": "Administrator",
"is_group": false,
"result": null,
"userdata": {},
"finished_timestamp": null,
"name": "36",
"events": {
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/36/events"
},
"state": "running",
"id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/35",
"created_timestamp": "2018-06-06T20:22:08Z",
"duration": 123.456,
"options": {
  "description": "Testing myplan",
  "plan_name": "myplan",
  "parameters": {
    "nodes": [
      "orchestrator.example.com"
    ],
    "environment": [],
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false,
    "result": null,
    "userdata": {}},
  "finished_timestamp": null,
  "name": "35",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/35/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/35",
  "created_timestamp": "2018-06-06T20:22:07Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",

"plan_name": "myplan",
"parameters": {
  "nodes": [
    "orchestrator.example.com"
  ]
},
"sensitive": [],
"environment": "dev",
"scheduled_job_id": null
},
"owner": {
  "email": "",
  "is_revoked": false,
  "last_login": "2018-06-06T20:22:06.327Z",
  "is_remote": false,
  "login": "admin",
  "is_superuser": true,
  "id": "42bf351c-f9ec-40af-84ad-e976feci7f4bd",
  "role_ids": [
    1
  ],
  "display_name": "Administrator",
  "is_group": false
},
"result": null,
"userdata": {}},
{
  "finished_timestamp": null,
  "name": "34",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/34/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/34",
  "created_timestamp": "2018-06-06T20:22:07Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
    "sensitive": [],
    "environment": "production",
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976feci7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false
  }
}
GET /plan_jobs/:job-id
List all the details of a given plan job.

Response format
The response is a JSON object that lists all details of a given plan job. The following keys are used:
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to the given plan job.</td>
</tr>
<tr>
<td>name</td>
<td>The ID of the plan job.</td>
</tr>
<tr>
<td>state</td>
<td>The current state of the plan job: running, success, or failure</td>
</tr>
<tr>
<td>options</td>
<td>Information about the plan job: description, plan_name, and any parameters.</td>
</tr>
<tr>
<td>description</td>
<td>The user-provided description for the plan job.</td>
</tr>
<tr>
<td>plan_name</td>
<td>The name of the plan that was run, for example package::install.</td>
</tr>
<tr>
<td>parameters</td>
<td>The parameters passed to the plan for the job.</td>
</tr>
<tr>
<td>result</td>
<td>The output from the plan job.</td>
</tr>
<tr>
<td>owner</td>
<td>The subject ID and login for the user that requested the job.</td>
</tr>
<tr>
<td>timestamp</td>
<td>The time when the plan job state last changed.</td>
</tr>
<tr>
<td>created_timestamp</td>
<td>The time the plan job was created.</td>
</tr>
<tr>
<td>finished_timestamp</td>
<td>The time the plan job finished.</td>
</tr>
<tr>
<td>events</td>
<td>A link to the events for a given plan job.</td>
</tr>
<tr>
<td>status</td>
<td>A hash of jobs that ran as part of the plan job, with associated lists of states and their enter and exit times.</td>
</tr>
<tr>
<td>userdata</td>
<td>An object of arbitrary key/value data supplied to the job.</td>
</tr>
</tbody>
</table>

For example:

```json
{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/1234",
  "name": "1234",
  "state": "success",
  "options": {
    "description": "This is a plan run",
    "plan_name": "package::install",
    "parameters": {
      "foo": "bar"
    }
  },
  "result": {
    "output": "test\n"
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "YYYY-MM-DDT17:06:48.170Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [1
    ],
    "display_name": "Administrator",
```

© 2022 Puppet, Inc., a Perforce company
Error responses

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the job-id in the request is not an integer, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-job</td>
<td>If the plan job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

Related information

Orchestrator API error responses on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Tasks endpoints

Use the /tasks endpoint to view details about the tasks pre-installed by PE and those you've installed.
GET /tasks
Lists all tasks in a given environment.

Parameters
The request accepts this query parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>Returns the tasks in the specified environment. If unspecified, defaults to production.</td>
</tr>
</tbody>
</table>

Response format
The response is a JSON object that lists each known task with a link to additional information, and uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>A map containing a name key specifying the environment's name and a code_id key indicating the code ID where the task is listed.</td>
</tr>
<tr>
<td>items</td>
<td>Contains an array of all known tasks.</td>
</tr>
<tr>
<td>id</td>
<td>An absolute URL where the task's details are listed.</td>
</tr>
<tr>
<td>name</td>
<td>The full name of the task.</td>
</tr>
</tbody>
</table>

```json
{
    "environment": {
        "name": "production",
        "code_id": "urn:puppet:code-id:1:a86da166c30f871823f9b2ea224796e834840676;production"
    },
    "items": [
        {
            "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/install",
            "name": "package::install"
        },
        {
            "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/upgrade",
            "name": "package::upgrade"
        },
        {
            "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/exec/init",
            "name": "exec"
        }
    ]
}
```

Error responses
For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the environment parameter is not a legal environment name, the server returns a 400 response.</td>
</tr>
</tbody>
</table>
**Key** | **Definition**
--- | ---
puppetlabs.orchestrator/unknown-environment | If the specified environment doesn't exist, the server returns a 404 response.

**Related information**

Orchestrator API error responses on page 760

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**GET /tasks/:module/:taskname**

Returns data about a specified task, including metadata and file information. For the default task in a module, :taskname is init.

**Parameters**

The request accepts this query parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>Returns the tasks in the specified environment. If unspecified, defaults to production.</td>
</tr>
</tbody>
</table>

**Response format**

The response is a JSON object that includes information about the specified task, and uses these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL where the task's details are listed.</td>
</tr>
<tr>
<td>name</td>
<td>The full name of the task.</td>
</tr>
<tr>
<td>environment</td>
<td>A map containing a name key specifying the environment's name and a code_id key indicating the code ID where the task is listed.</td>
</tr>
<tr>
<td>metadata</td>
<td>A map containing the contents of the &lt;task&gt;.json file.</td>
</tr>
<tr>
<td>files</td>
<td>An array of the files in the task.</td>
</tr>
<tr>
<td>filename</td>
<td>The base name of the file.</td>
</tr>
<tr>
<td>uri</td>
<td>A map containing path and params fields to construct a URL to download the file content. The client determines which host to download the file from.</td>
</tr>
<tr>
<td>sha256</td>
<td>The SHA-256 hash of the file content, in lowercase hexadecimal form.</td>
</tr>
<tr>
<td>size_bytes</td>
<td>The size of the file content in bytes.</td>
</tr>
</tbody>
</table>

For example:

```json
{
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/install",
    "name": "package::install",
    "environment": {
        "name": "production",
```
Error responses

For this endpoint, the kind key of the error displays the conflict.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppetlabs.orchestrator/validation-error</td>
<td>If the environment parameter is not a legal environment name, or the module or taskname is invalid, the server returns a 400 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-environment</td>
<td>If the specified environment doesn’t exist, the server returns a 404 response.</td>
</tr>
<tr>
<td>puppetlabs.orchestrator/unknown-task</td>
<td>If the specified task doesn’t exist within the specified environment, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

Related information

Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**Root endpoints**

Use the `/orchestrator` endpoint to return metadata about the orchestrator API.

**GET /orchestrator**

A request to the root of the Puppet orchestrator returns metadata about the orchestrator API, along with a list of links to application management resources.

**Response format**

Responses use the following format:

```json
{
   "info": {
      "title": "Application Management API (EXPERIMENTAL)",
      "description": "Multi-purpose API for performing application management operations",
      "warning": "This version of the API is experimental, and might change in backwards-incompatible ways in the future",
      "version": "0.1",
      "license": {
         "name": "Puppet Enterprise License",
         "url": "https://puppetlabs.com/puppet-enterprise-components-licenses"
      }
   },
   "status": {
      "name": "status",
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/status"
   },
   "collections": [
      {
         "name": "environments",
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/environments"
      },
      {
         "name": "jobs",
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs"
      }
   ],
   "commands": [
      {
         "name": "deploy",
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/command/deploy"
      },
      {
         "name": "stop",
         "id": "https://orchestrator.example.com:8143/orchestrator/v1/command/stop"
      }
   ]
}
```

**Error responses**

For this endpoint, the server returns a 500 response.

**Related information**

[Orchestrator API error responses](#) on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

**Usage endpoints**

Use the `/usage` endpoint to view details about the active nodes in your deployment.

**GET /usage**

List the daily node usage of the orchestrator and nodes that are present in PuppetDB.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_date</td>
<td>The first day to include in the result, in the format YYYY-MM-DD</td>
</tr>
<tr>
<td>end_date</td>
<td>The last day to include in the result, in the format YYYY-MM-DD. The date must be &gt;= start_date.</td>
</tr>
<tr>
<td>events</td>
<td>Specifies whether to include or exclude the daily event details from the response. Defaults to include. When set to exclude, the keys corrective_agent_changes, intentional_agent_changes, nodes_affected_by_task_runs, and nodes_affected_by_plan_runs are removed.</td>
</tr>
</tbody>
</table>

**Response format**

The response is a JSON object indicating the total number of active nodes and subtotals of the nodes with and without an agent installed.

The following keys are used:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Contains an array entries from most recent to least recent.</td>
</tr>
<tr>
<td>date</td>
<td>An ISO 8601 date representing the date of the request in UTC.</td>
</tr>
<tr>
<td>total_nodes</td>
<td>The total number of nodes used since UTC midnight.</td>
</tr>
<tr>
<td>nodes_with_agent</td>
<td>The number of nodes in pdb at the time the request is made.</td>
</tr>
<tr>
<td>nodes_without_agent</td>
<td>The number of unique nodes that don't have an agent that were used since UTC midnight.</td>
</tr>
<tr>
<td>corrective_agent_changes</td>
<td>The number of corrective changes made by agent runs.</td>
</tr>
<tr>
<td>intentional_agent_changes</td>
<td>The number of intentional changes made by agent runs.</td>
</tr>
<tr>
<td>nodes_affected_by_task_runs</td>
<td>The number of tasks run (counted per node that a task runs on).</td>
</tr>
<tr>
<td>nodes_affected_by_plan_runs</td>
<td>The number of plans run (counted per node that a plan runs on).</td>
</tr>
</tbody>
</table>
An optional object that includes information about the original query.

The starting day for the request (if specified).

The last day requested (if specified).

For example:

```json
{
  "items": [
    {
      "date": "2018-06-08",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-07",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-06",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-05",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    }
  ],
  "pagination": {
    "start_date": "2018-06-01",
    "end_date": "2018-06-30"
  }
}
```

When the `events` parameter is set to `exclude`, the response excludes the keys that specify Puppet events activity and only includes information about node counts.

For example:

```json
{
  "items": [
    {
      "date": "2018-06-08",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-07",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-06",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    },
    {
      "date": "2018-06-05",
      "total_nodes": 100,
      "nodes_with_agent": 95,
      "nodes_without_agent": 5
    }
  ]
}
```
Error Responses
For this endpoint, the kind key of the error displays the conflict.

Related information
Orchestrator API error responses on page 760
From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Scopes endpoints
Use the scopes endpoint to retrieve resources for task constraints.

GET /scopes/task_targets
List all the known task targets (set of tasks + set of nodes + set of node groups) stored in orchestrator.

Response format
The response is a JSON object that lists the known task targets and uses the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Contains an array of all the known task targets.</td>
</tr>
</tbody>
</table>

The following keys are included under items:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to retrieve the individual task target.</td>
</tr>
<tr>
<td>name</td>
<td>A string that is a unique identifier for the task target.</td>
</tr>
<tr>
<td>display_name</td>
<td>An optional string value that can be used to describe the task target.</td>
</tr>
<tr>
<td>tasks</td>
<td>An optional array of tasks that this task target corresponds to.</td>
</tr>
<tr>
<td>all_tasks</td>
<td>An optional boolean to determine if all tasks can be run on designated targets.</td>
</tr>
<tr>
<td>nodes</td>
<td>An array of certnames that this task target corresponds to. The array can be empty.</td>
</tr>
<tr>
<td>node_groups</td>
<td>An array of node group ids that this task target corresponds to. The array can be empty.</td>
</tr>
<tr>
<td>pql_query</td>
<td>(optional) A string that is a single PQL query that the task target corresponds to.</td>
</tr>
</tbody>
</table>
For example:

```json
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/1",
      "name": "1",
      "tasks": [
        "package::install",
        "exec"
      ],
      "all_tasks": "false",
      "nodes": [
        "wss6c3w9wngpycg",
        "jjj2h5w8gpycgwn"
      ],
      "node_groups": [
        "3c4df64f-7609-4d3l-9c2d-acfa52ed66ec",
        "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
      ],
      "pql_query": "nodes[certname] { catalog_environment = "$production" }
    }
    ,
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/2",
      "name": "2",
      "tasks": [
        "imaginary::task"
      ],
      "all_tasks": "false",
      "nodes": [
        "mynode"
      ],
      "node_groups": []
    }
    ,
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/3",
      "name": "3",
      "all_tasks": true,
      "nodes": [
        "xxx6c3w9wngpycg",
        "bbb2h5w8gpycgwn"
      ],
      "node_groups": [
        "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
        "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
      ]
    }
  ]
}
```

**GET /scopes/task_targets/:id**

Get information about a specific task_target.

**Response format**

The response is a JSON object that provides the details about the task_target and uses the following keys:
<table>
<thead>
<tr>
<th>Keys</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An absolute URL to retrieve the individual task_target.</td>
</tr>
<tr>
<td>name</td>
<td>A string that is a unique identifier for the task_target.</td>
</tr>
<tr>
<td>display_name</td>
<td>An optional string value that can be used to describe the task_target.</td>
</tr>
<tr>
<td>tasks</td>
<td>An optional array of tasks that this task_target corresponds to.</td>
</tr>
<tr>
<td>all_tasks</td>
<td>An optional boolean to determine if all tasks can be run on designated targets.</td>
</tr>
<tr>
<td>nodes</td>
<td>An array of certnames that this task_target corresponds to. The array can be empty.</td>
</tr>
<tr>
<td>node_groups</td>
<td>An array of node group ids that this task_target corresponds to. The array can be empty.</td>
</tr>
<tr>
<td>pql_query</td>
<td>(optional) A string that is a single PQL query that the task_target corresponds to.</td>
</tr>
</tbody>
</table>

For example:

```
{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/1",
  "name": "1",
  "tasks": [
    "package::install",
    "exec"
  ],
  "all_tasks": "false",
  "nodes": [
    "wss6c3w9wnpypycg",
    "jjj2h5w8gpypycgw"
  ],
  "node_groups": [
    "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
    "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
  ],
  "pql_query": "nodes[certname] { catalog_environment = "production" }"
}
```

**Error responses**

For this endpoint, the `kind` key of the error displays the conflict. See the Orchestrator API error responses on page 760 documentation for the general format of error responses.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>:puppetlabs.orchestrator/unknown-task-target</td>
<td>If the specified id doesn't exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>
Orchestrator API error responses

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Every error response from the Puppet orchestrator is a JSON response. Each response is an object that contains the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>The kind of error encountered.</td>
</tr>
<tr>
<td>msg</td>
<td>The message associated with the error.</td>
</tr>
<tr>
<td>details</td>
<td>A hash with more information about the error.</td>
</tr>
</tbody>
</table>

For example, if an environment does not exist for a given request, an error is raised similar to the following:

```json
{
    "kind": "puppetlabs.orchestrator/unknown-environment",
    "msg": "Unknown environment doesnotexist",
    "details": {
        "environment": "doesnotexist"
    }
}
```

Migrating Bolt tasks and plans to PE

If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a Puppet Enterprise (PE) environment. This lets you manage and run tasks and plans using PE and the console. Bolt projects have the same structure as Puppet modules, and they can be loaded from the modules directory of a PE environment.

The control repo is a central Git repository from which PE fetches content. An environment is a space for PE authors to write and install content, similar to a Bolt project.

There are two ways to get your Bolt content into an environment:

- Move your Bolt code to a new control repo. Do this if you have a Boltdir, or an embedded project directory, in a repo that also contains other code that you do not want to migrate to PE.
- Configure PE to point to the Bolt project. Do this is if you have a dedicated repo for Bolt code, or a local project directory, and don't want to duplicate it in PE.

Move Bolt content to a new PE repo

Move your Bolt project content out of your Boltdir and into a fresh PE control repo.

Before you begin

- Install PE on your machine. See Getting started with Puppet Enterprise on page 157.
- Set up your PE control repo and environments. See Managing environments with a control repository on page 764.

To move Bolt content to a repo:
1. Commit the contents of your Bolt project to a branch of your PE control repo. Place the Bolt project under the modules directory. If you're using Bolt module workflows, make sure you run `bolt module install` and commit the resulting Puppetfile to your control repo.

Your new structure is similar to a project directory in Bolt, for example:

```
test-environment/
  ### Puppetfile
  ### bolt-project.yaml
  ### data
  #   ### common.yaml
  ### inventory.yaml
  ### modules
    ### project
    ### manifests
      #   ### my_class.pp
    ### plans
      #   ### deploy.pp
      #   ### diagnose.pp
    ### tasks
      ### init.json
      ### init.py
```

2. Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment in PE.

3. Add the modulepath setting to the `environment.conf` file by adding the following line:

```
modulepath = modules:modules:$basemodulepath
```

**Note:** PE picks up modules only from the `modules` directory. It's important to add modules to the modulepath setting so it matches the defaults for your Bolt project. If you have a modulepath setting in `bolt-project.yaml`, match it to the modulepath setting in `environment.conf`.

4. Publish the branch to the PE control repo.

5. Deploy code using `puppet code deploy --<ENVIRONMENT>`, where `<ENVIRONMENT>` is the name of your branch, to commit the new branch to Git.

**Note:** You can also deploy code using a webhook. See Triggering Code Manager with a webhook on page 798 for more information.

After you deploy code, modules (and the tasks and plans within them) listed in the new environment's Puppetfile are available to use in PE.

**Related information**

Plans in PE versus Bolt plans on page 664
Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

**Point PE to a Bolt project**

Allow PE to manage content in your dedicated Bolt repo.

**Before you begin**

- Install PE on your machine. See Getting started with Puppet Enterprise on page 157.
- Ensure your Bolt project follows the local project directory structure.

To point PE to your Bolt content:
1. To allow access to the control repo, generate a private SSH key without a password:
   a) To generate the key pair, run:
   ```sh
   ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
   ```
   b) To allow the pe-puppet user to access the key, run:
   ```sh
   puppet infrastructure configure
   ```
   Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.
   c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as GitHub, BitBucket Server, or GitLab).
   **Important:** Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

2. Change the name of your branch to `production`. PE uses branches in Git as environments and the default environment is `production`.

3. Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment.

4. Add the `modulepath` setting to the `environment.conf` file by adding the following line:
   ```sh
   modulepath = modules:modules:$basemodulepath
   ```
   **Note:** PE picks up modules only from the `modules` directory. It's important to add modules to the `modulepath` setting so it matches the defaults for your Bolt project. If you have a `modulepath` setting in `bolt-project.yaml`, match it to the `modulepath` setting in `environment.conf`.

5. Publish the branch to the PE control repo.

6. Deploy code using `puppet code deploy --<ENVIRONMENT>`, where `<ENVIRONMENT>` is the name of your branch, to commit the new branch to Git.
   **Note:** You can also deploy code using a webhook. See Triggering Code Manager with a webhook on page 798 for more information.

After you deploy code, modules (and the tasks and plans within them) listed in the new environment's Puppetfile are available to use in PE.

**Related information**

**Plans in PE versus Bolt plans** on page 664
Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

**PE workflows for Bolt users**
Understand the differences between PE and Bolt commands and workflows before you start running tasks and plans in PE.

**Connecting to nodes**
You must connect PE to each node you want to run tasks on or include in a plan. See Add nodes to the inventory on page 160 for instructions on adding agent or agentless nodes to your inventory.
Installing tasks and plans

In PE, as in Bolt, you use the `mod` command to download modules. But instead of running the `bolt puppetfile install` command to install them, you trigger Code Manager and deploy code using the `puppet code deploy` command. See Triggering Code Manager on the command line on page 792.

Running tasks and plans

PE does not recognize the `bolt` command for running tasks and plans. Instead, use the `puppet task run` and `puppet plan run` commands, or use the console.

To run tasks or plans from the command line, see:
- Running tasks from the command line on page 641
- Running plans from the command line on page 668

To run tasks or plans from the console, see:
- Running tasks from the console on page 635
- Running plans from the console on page 667

Limitations in PE

Not everything in Bolt works in PE. For example, many pre-installed Bolt modules are not included in PE and many plan functions do not work, such as `file::exists` and `set_feature`. See Plans in PE versus Bolt plans on page 664.

Managing and deploying Puppet code

Puppet Enterprise (PE) includes built-in tools for managing and deploying your Puppet code. Code Manager and r10k are code management tools that automatically install modules, create and maintain environments, and deploy new code to your primary server and compilers, all based on version control of your Puppet code and data.

You can use either Code Manager or r10k to manage and deploy Puppet code. Both tools are built into PE and do not require separate installations.

Code Manager is the recommended tool for managing Puppet code in PE. Code Manager automates the deployment of your Puppet code and data. You make code and data changes on your workstation, push changes to your Git repository, and, from there, Code Manager creates environments, installs modules, and deploys the new code to your primary server and compilers, without interrupting agent runs.

If you are unable to use Code Manager, you can use r10k to manage your code.

- Managing environments with a control repository on page 764
To manage your Puppet code and data with Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.
- Managing environment content with a Puppetfile on page 768
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.
- Managing code with Code Manager on page 774
Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.
- Managing code with r10k on page 816
r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to manually deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.
Managing environments with a control repository

To manage your Puppet code and data with Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

How the control repository works

Code management relies on version control to track, maintain, and deploy your Puppet code and data. The control repository (or control repo) is a Git repository that code management uses to manage environments in your infrastructure. When you update code and data in your control repo, code management updates your environments accordingly.

Code management creates and maintains environments based on the branches in your control repo. For example, if your control repo has a production branch, a development branch, and a testing branch, code management creates a production environment, a development environment, and a testing environment. Each environment has its own version of your Puppet code and data based on the contents of the corresponding branch.

Environments are created on the primary server at /etc/puppetlabs/code/environments. You can learn more About environments in the Puppet documentation.

CAUTION: When you enable code management, Puppet manages the environment directories and does not preserve existing environments. Existing environments with the same names as new ones are overwritten, and environments not represented in the control repo are erased. If you were already using environment directories, make sure you commit those files or code to the corresponding branches of your control repo (or back them up elsewhere) before you start configuring code management.

At minimum, a control repo includes:

• A Git remote repository. This is where your control repo is stored on your version control host.
• A default branch named production, rather than the usual Git default of master. You might have additional branches for other environments, such as development or testing.
• A Puppetfile to manage your environment content.
• An environment.conf file that modifies the $modulepath setting to allow environment-specific modules and settings.

There are two ways to create control repos. To ensure your control repo has the recommended structure, code examples, and configuration scripts, Create a control repository from the Puppet template on page 161. This template covers most customer situations. If you cannot access the internet or cannot download modules directly from the Forge because of your organization’s security rules, Create an empty control repo on page 766 and add the necessary files to it.

Restriction: For Windows systems, make sure your version control is configured to use CRLF line endings. Check your version control host's documentation for instructions on how to do this.

It is possible to have multiple control repos, and you can use separate repos to contain your module content or other data. If you have multiple repos, you need to:

• Customize your Git settings (particularly the repositories setting).
  • Configuring Code Manager Git settings
  • Configuring r10k Git settings on page 822
• Map your sources.
  • Configuring Code Manager sources
  • Configuring r10k sources
Create a control repository from the Puppet template

To create a control repository (or control repo) that has the recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

The Puppet control repo template contains the necessary files to configure a functioning code management control repo plus helpful Puppet code examples, including:

- Basic code examples for setting up roles and profiles.
- A Puppetfile that references modules to manage content in your environments.
- An example Hiera configuration file and hieradata directory.
- A config_version script that tells you which version of code from your control repo was applied to your agents.
- An environment.conf file that implements the config_version script and a site-modules directory for roles, profiles, and custom modules.

In situations where you can't access the internet, or where organizational security policies prevent downloading modules from the Forge, you can Create an empty control repo on page 766 and add the necessary files to it.

To use the template, you must set up a private SSH key, copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

**Important:** The following steps assume you are using GitHub Enterprise with SSH. For more information and instructions for other version control hosts, such as GitLab or BitBucket, go to the Puppet control-repo template README.

1. To allow access to the control repo, generate a private SSH key without a password:
   a) To generate the key pair, run:

   ```
   ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
   ```

   b) To allow the pe-puppet user to access the key, run:

   ```
   puppet infrastructure configure
   ```

   Your private key is located at /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519, and your public key is at /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub.

   c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as GitHub, GitLab Server, or GitLab).

   **Important:** Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

2. In your Git user account or organization, create a repository named control-repo, and make sure a README is not automatically generated when you create the repo. Take note of the repo’s SSH URL.

   **Important:** Do not use an existing repo. The template requires a new, empty SSH URL.

3. If you haven't already installed Git, run `yum install git`.

4. To clone the Puppet control-repo template, run:

   ```
   git clone https://github.com/puppetlabs/control-repo.git
   ```

5. Change to the control-repo directory: `cd control-repo`

6. Remove the template repo as the origin: `git remote remove origin`
7. Set your control repo as the origin: `git remote add origin <SSH_URL_FOR_YOUR_CONTROL_REPO>`

8. Push the contents of the production branch of the cloned control repo to your remote control repo: `git push origin production`

You now have a control repository based on the Puppet control-repo template. After configuring Code Manager, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, Code Manager detects and deploys your infrastructure changes.

By using the control-repo template, you now also have a Puppetfile to which you can add and manage content, like module code.

**Related information**

- Managing environment content with a Puppetfile on page 768
  A Puppetfile specifies detailed information about each environment’s Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

- Managing code with Code Manager on page 774
  Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

- Add an environment on page 768
  Create new environments by creating branches based on your control repo’s production branch.

**Create an empty control repo**

In situations where you can’t access the internet, or where organizational security policies prevent downloading modules from the Forge, you can create an empty control repo and add the necessary files to it.

When you can’t use the Puppet control repo template, you must create a new repo on your Git host, clone it to your workstation, make changes to the repo (such as adding a configuration file to allow code management tools to find modules in your module directories), and push your changes to the remote repo on your Git host.

1. To allow access to the control repo, generate a private SSH key without a password:
   a) To generate the key pair, run:

```
ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

   b) To allow the pe-puppet user to access the key, run:

```
puppet infrastructure configure
```

   Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.

   c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host’s documentation (such as GitHub, BitBucket Server, or GitLab).

   **Important:** Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.
2. In your Git account, create a repository with the name you want your control repo to have (we recommend control-repo), and take note of the repo's SSH URL. Check your Git host's documentation for exact instructions, because this process varies for each host. For example, to create a new repo on GitHub:
   a. Click + at the top of the page, and choose New repository.
   b. Select the account Owner for the repository.
   c. Name the repository (for example, control-repo).
   d. Note the repository's SSH URL for later use.

   **Tip:** While you can use an existing repo as your control repo, we recommend starting with a new repo to avoid possible unexpected changes to existing files and directories once you enable code management.

3. Clone the new repo to your workstation: `git clone <REPOSITORY_URL>`

4. In the control repo's main directory, create a configuration file named `environment.conf`.
   The `environment.conf` file allows code management tools to find modules in your site- and environment-specific module directories. You can learn more about this file and its contents in the Puppet `environment.conf` documentation.

5. To set the module path, open the `environment.conf` file in a text editor, add the following line, and then save and close the file.

   ```text
   modulepath=site-modules:modules:$basemodulepath
   ```

6. Add the new file to the index and commit your change by running `git add environment.conf` and then `git commit -m "add environment.conf"`

7. Rename the master branch to production by running `git branch -m master production`

   **Important:** Puppet Enterprise requires the control repo's default branch to be production.

8. Push your repository's production branch from your workstation to your Git host by running `git push -u origin production`

After configuring the Puppetfile and code management, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, code management detects and deploys your infrastructure changes.

After creating your control repo, you must create a Puppetfile to manage your environment content with code management. Then, you must configure either Code Manager (recommended) or r10k.

**Related information**

- [Managing environment content with a Puppetfile](#)
- [Managing code with Code Manager](#)
- [Managing code with r10k](#)

**Managing environment content with a Puppetfile**
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

**Managing code with Code Manager**
Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

**Managing code with r10k**
r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to manually deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

**Add an environment**

© 2022 Puppet, Inc., a Perforce company
Create new environments by creating branches based on your control repo's production branch.

**Add an environment**
Create new environments by creating branches based on your control repo's production branch.

**Before you begin**
You must have:
- A control repository.
- A Puppetfile in your control repo's production branch.
- Configured Code Manager or r10k.
- Selected a code management deployment method (either the puppet-code command or a webhook).

**Restriction:** If you have multiple control repos, you can't repeat branch names unless you use a source prefix. Go to Configuring sources on page 787 for more information.

1. In your control repo, create a new branch based on the production branch:
   ```bash
git branch <NEW_BRANCH_NAME>
   ```
2. Check out the new branch:
   ```bash
git checkout <NEW_BRANCH_NAME>
   ```
3. Edit the Puppetfile to track the necessary modules and data for your new environment, and then save your changes.
4. Commit your changes:
   ```bash
git commit -m "prepare Puppetfile for new environment"
   ```
5. Push your changes:
   ```bash
git push origin <NEW_BRANCH_NAME>
   ```
6. Deploy your environments as you normally would, either on the command line or with a webhook.

Code management detects the new environment in your control repo and begins managing it, as explained in How the control repository works on page 764.

**Delete an environment from code management**
To delete an environment that is being managed by Code Manager or r10k, delete the corresponding branch from your control repository.

1. On your control repo's production branch, delete the environment's corresponding remote branch by running
   ```bash
git push origin --delete <BRANCH_TO_DELETE>
   ```
2. Delete the local branch by running
   ```bash
git branch -d <BRANCH_TO_DELETE>
   ```
3. Deploy your environments as you normally would, either on the command line or with a webhook.

**Important:** If you use webhooks to deploy environments, Code Manager deletes the environment from the primary server's live code directories the next time it deploys changes to any other environment. If you want to immediately delete the environment from the primary server's live code directories, deploy all environments manually by running
   ```bash
puppet-code deploy --all --wait
   ```

---

**Managing environment content with a Puppetfile**
A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Both Code Manager and r10k use a Puppetfile to install and manage the content of your environments.
The Puppetfile

The Puppetfile specifies the modules and data that you want in each environment. The Puppetfile can specify what version of modules you want, how the modules and data are to be loaded, and where they are placed in the environment.

A Puppetfile is a formatted text file that specifies the modules and data that you want brought into your control repo. Typically, a Puppetfile controls content such as:

• Modules from the Forge
• Modules from Git repositories
• Data from Git repositories

For each environment that you want to manage content in, you need a Puppetfile. Create a base Puppetfile in your default environment (usually production). As you create new branches based on your default branch, each environment inherits this base Puppetfile. You can then edit each environment's Puppetfile as needed.

Managing modules with a Puppetfile

With code management, install and manage your modules only with a Puppetfile.

Almost all Puppet manifests are kept in modules, collections of Puppet code and data with a specific directory structure. By default, Code Manager and r10k install content in a modules directory (.modules) in the same directory the Puppetfile is in. For example, declaring the puppetlabs-apache module in the Puppetfile normally installs the module into ./modules/apache. To learn more about modules, see the module documentation.

Important:

With Code Manager and r10k, do not use the puppet module command to install or manage modules. Instead, code management depends on the Puppetfile in each of your environments to install, update, and manage your modules. If you’ve installed modules to the live code directory with puppet module install, Code Manager deletes them.

The Puppetfile does NOT include Forge module dependency resolution. You must make sure that you have every module needed for all of your specified modules to run. In addition, Forge module symlinks are unsupported; when you install modules with r10k or Code Manager, symlinks are not installed.

Including your own modules

If you develop your own modules, maintain them in version control and include them in your Puppetfile as you would declare any module from a repository. If you have content in your control repo's module directory that is not listed in your Puppetfile, code management purges it. (The control repo module directory is, by default, ./modules relative to the location of the Puppetfile.)

Deploying code

When you install or update a module, you must trigger Code Manager or r10k to deploy the new or updated code to your environments.

Note: Code Manager does not automatically deploy modules' spec directories. These directories are for testing only, and they are not useful in a production environment. If you would like to include a module's spec directory when deploying, add exclude_spec: false to the module declaration in your Puppetfile.

With Code Manager, you can deploy code on the command line or using a webhook:

• Triggering Code Manager on the command line on page 792
• Triggering Code Manager with a webhook on page 798

With r10k, you can deploy code using the command line:

• Deploying environments with r10k on page 825
Creating a Puppetfile

Your Puppetfile manages the content you want to maintain in that environment.

In a Puppetfile, you can declare:

- Modules from the Forge.
- Modules from a Git repository.
- Data or other non-module content (such as Hiera data) from a Git repository.

You can declare any or all of this content as needed for each environment. Each module or repository is specified with a `mod` directive, along with the name of the content and other information the Puppetfile needs so that it can correctly install and update your modules and data.

It's best to create your first Puppetfile in your production branch. Then, as you create branches based on your production branch, edit each branch's Puppetfile as needed.

Create a Puppetfile

Create a Puppetfile that manages the content maintained in your environment.

**Before you begin**

Set up a control repo, with `production` as the default branch. To learn more about control repositories, see the control repository documentation.

Create a Puppetfile in your production branch, and then edit it to declare the content in your production environment with the `mod` directive.

1. On your production branch, in the root directory, create a file named `Puppetfile`.

2. In a text editor, for example Visual Studio Code (VS Code), edit the Puppetfile, declaring modules and data content for your environment. Note that Puppet has an extension for VS Code that supports syntax highlighting of the Puppet language.

   You can declare modules from the Forge or you can declare Git repositories in your Puppetfile. See the related topics about declaring content in the Puppetfile for details and code examples.

Configure Code Manager or r10k.

Change the Puppetfile module installation directory

If needed, you can change the directory to which the Puppetfile installs all modules.

To specify a module installation path other than the default modules directory (`./modules`), use the `moduledir` directive.

This directive applies to all content declared in the Puppetfile. You must specify this as a relative path at the top of the Puppetfile, **before** you list any modules.

To change the installation paths for only certain modules or data, declare those content sources as Git repositories and set the `install_path` option. This option overrides the `moduledir` directive. See the related topic about how to declare content as a Git repo for instructions.

Add the `moduledir` directive at the top of the Puppetfile, specifying your module installation directory relative to the location of the Puppetfile.

```
moduledir 'thirdparty'
```

Declare Forge modules in the Puppetfile

Declare Forge modules in your Puppetfile, specifying the version and whether or not code management keeps the module updated.

Specify modules by their full name. You can specify the most recent version of a module, with or without updates, or you can specify a specific version of a module.
Note: Some existing Puppetfiles contain a forge setting that provides legacy compatibility with librarian-puppet. This setting is non-operational for Code Manager and r10k. To configure how Forge modules are downloaded, specify forge_settings in Hiera instead. See the topics about configuring the Forge settings for Code Manager or r10k for details.

In your Puppetfile, specify the modules to install with the mod directive. For each module, pass the module name as a string, and optionally, specify what version of the module you want to track.

If you specify no options, code management installs the latest version and keeps the module at that version. To keep the module updated, specify :latest. To install a specific version of the module and keep it at that version, specify the version number as a string.

```plaintext
mod 'puppetlabs/apache'
mod 'puppetlabs/ntp', :latest
mod 'puppetlabs/stdlib', '0.10.0'
```

This example:
- Installs the latest version of the apache module, but does not update it.
- Installs the latest version of the ntp module, and updates it when environments are deployed.
- Installs version 0.10.0 of the stdlib module, and does not update it.

Declare Git repositories in the Puppetfile

List the modules, data, or other non-module content that you want to install from a Git repository.

To specify any environment content as a Git repository, use the mod directive with the :git option. This is useful for modules you don't get from the Forge, such as your own modules, as well as data or other non-module content.

To install content and keep it updated to the master branch, declare the content name and specify the repository with the :git directive. Optionally, specify an :install_path for the content.

```plaintext
mod 'apache',
    :git => 'https://github.com/puppetlabs/puppetlabs-apache'
mod 'site_data',
    :git => 'git@git.example.com:site_data.git',
    :install_path => 'hieradata'
```

This example installs the apache module and keeps that module updated with the master branch of the listed repository. It also installs site data content from a Git repository into the environment's ./hieradata/site_data subdirectory.

Note: Content is installed in the modules directory and treated as a module, unless you use the :install_path option. Use this option with non-module content to keep your data separate from your modules.

Specify installation paths for repositories

You can set individual installation paths for any of the repositories that you declare in the Puppetfile.

The :install_path option allows you to separate non-module content in your directory structure or to set specific installation paths for individual modules. When you set this option for a specific repository, it overrides the moduledir setting.

To install the content into a subdirectory in the environment, specify the directory with the install_path option.
To install into the root of the environment, specify an empty value.

```plaintext
mod 'site_data_1',
    :git => 'git@git.example.com:site_data_1.git',
    :install_path => 'hieradata'
mod 'site_data_2',
    :git => 'git@git.example.com:site_data_2.git',
```
This example installs site data content from the `site_data_1` repository into `./hieradata/site_data` and content from `site_data_2` into `./site_data` subdirectory.

### Declare module or data content with SSH private key authentication

To declare content protected by SSH private keys, declare the content as a repository, and then configure the private key setting in your code management tool.

1. Declare your repository content, specifying the Git repo by the SSH URL.

   ```
   mod 'myco/privatemod',
   :git => 'git@git.example.com:myco/privatemod.git'
   ```

   **Note:** If modifying the Puppetfile triggers a code deployment, expect the code deployment to fail. You must complete the next step to get a successful code deployment.

2. Configure the private key settings by modifying the following Code Manager or r10k parameters in Hiera:

   - To set a key for all Git operations, use the private key setting under `git-settings`.
   - To set a private key for an individual remote, set the private key in the `repositories` hash in `git-settings` for each specific remote.

   For more information about these parameters, refer to Configuring Git settings on page 786 for Code Manager or Configuring r10k Git settings on page 822 for r10k.

   To make these changes, you must follow the steps described in Customize Code Manager configuration in Hiera on page 784 and Customizing r10k configuration on page 818.

   After completing both steps, you might need to manually trigger a code deployment.

   - Triggering Code Manager on the command line on page 792
   - Triggering Code Manager with a webhook on page 798
   - Triggering Code Manager with custom scripts on page 799
   - Deploying environments with r10k on page 825

### Keep repository content at a specific version

The Puppetfile can maintain repository content at a specific version.

To specify a particular repository version, declare the version you want to track with your choice of the following options. Setting one of these options maintains the repository at that version and deploys any updates you make to that particular version.

- **ref**: Specifies the Git reference to check out. This option can reference either a tag, a commit, or a branch.
- **tag**: Specifies the repository by a certain tag value.
- **commit**: Specifies the repository by a certain commit.
- **branch**: Specifies a certain branch of the repository.
- **default_branch**: Specifies a default branch to use for deployment if the specified ref, tag, commit, or branch cannot be deployed. You must also specify one of the other version options. This is useful if you are tracking a relative branch of the control repository.

In the Puppetfile, declare the content, specifying the repository and version you want to track.

To install `puppetlabs/apache` and specify the '0.9.0' tag, use the `tag` option.

```
mod 'apache',
   :git => 'https://github.com/puppetlabs/puppetlabs-apache',
   :tag => '0.9.0'
```
To install puppetlabs/apache and use the branch option to track the ‘proxy_match’ branch.

```
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache',
  :branch => 'proxy_match'
```

To install puppetlabs/apache and use the commit option to track the ‘8df51aa’ commit.

```
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache',
  :commit => '8df51aa'
```

**Declare content from a relative control repo branch**

The branch option also has a special :control_branch option, which allows you to deploy content from a control repo branch relative to the location of the Puppetfile.

Normally, branch tracks a specific named branch of a repo, such as testing. If you set it to :control_branch, it instead tracks whatever control repo branch the Puppetfile is in. For example, if your Puppetfile is in the production branch, content from the production branch is deployed; if a duplicate Puppetfile is located in testing, content from testing is deployed. This means that as you create new branches, you don’t have to edit the inherited Puppetfile as extensively.

**Note:** The module repository branch names must match the control repository branch names in order to use :control_branch.

To track a branch within the control repo, declare the content with the :branch option set to :control_branch.

```
mod 'hieradata',
  :git => 'git@git.example.com:organization/hieradata.git',
  :branch => :control_branch
```

**Set a default branch for content deployment**

Set a default_branch option to specify what branch code management deploys content from if the given option for your repository cannot be deployed.

If you specified a ref, tag, commit, or branch option for your repository, and it cannot be resolved and deployed, code management can instead deploy the default_branch. This is mostly useful when you set branch to the :control_branch value.

If your specified content cannot be resolved and you have not set a default branch, or if the default branch cannot be resolved, code management logs an error and does not deploy or update the content.

In the Puppetfile, in the content declaration, set the default_branch option to the branch you want to deploy if your specified option fails.

```
# Track control branch and fall-back to master if no matching branch.
mod 'hieradata',
  :git => 'git@git.example.com:organization/hieradata.git',
  :branch => :control_branch,
  :default_branch => 'master'
```
Managing code with Code Manager

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

- **How Code Manager works** on page 774
  To automatically manage your environments and modules, Code Manager uses r10k and the file sync service to stage, commit, and sync your code.
- **Set up Code Manager** on page 777
  You must set up Code Manager to use it as your code management tool.
- **Configure Code Manager** on page 777
  To configure Code Manager, first enable Code Manager in Puppet Enterprise (PE), then set up authentication, and test the communication between the control repository and Code Manager.
- **Lockless code deploys** on page 781
  When deploying code, Puppet Server blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. However, you can enable lockless code deploys to deploy code without blocking requests to Puppet Server. With lockless code deploys, the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.
- **Customize Code Manager configuration in Hiera** on page 784
  To customize your Code Manager configuration, set parameters with Hiera.
- **Triggering Code Manager on the command line** on page 792
  The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.
- **Triggering Code Manager with a webhook** on page 798
  To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. The webhook is the simplest way to trigger Code Manager.
- **Triggering Code Manager with custom scripts** on page 799
  Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repositories, or custom notifications.
- **Troubleshooting Code Manager** on page 801
  Code Manager requires coordination between multiple components, including r10k and the file sync service. If you have issues with Code Manager, check that these components are functioning.
- **Code Manager API** on page 804
  Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

How Code Manager works

To automatically manage your environments and modules, Code Manager uses r10k and the file sync service to stage, commit, and sync your code.

Code Manager requires **Managing environments with a control repository** on page 764. You must create a control repository with branches for each environment that you want to create (such as production, development, or testing). Each branch must have a Puppetfile specifying exactly which modules to install in each environment. You can learn more about Environments in the Puppet documentation.

Code Manager creates directory environments based on the branches you've set up. Your control repository lives on a Git server, and this is where you push code that you want Code Manager to deploy.

There are three ways to trigger Code Manager to start a code deployment:

- A webhook from your Git server automatically starts the code deployment when you push code to the control repo.
- **Continuous Delivery for Puppet Enterprise** sends a deploy request to Code Manager.
- You use `puppet-code` to manually trigger Code Manager from the command line or with a custom script.
Once triggered, Code Manager uses r10k to fetch code from the Git server and places it into the staging directory (at /etc/puppetlabs/code-staging) on the primary server. Next, the file sync storage service on the primary server detects the change, and the file sync clients pause the Puppet Server to avoid conflicts during synchronization. Finally, the file sync clients synchronize the new code to the live code directories on the primary server and compilers (usually at /etc/puppetlabs/code). The following diagram illustrates this code deployment process.

**Related information**
- Triggering Code Manager on the command line on page 792
  The puppet-code command allows you to trigger Code Manager from the command line to deploy your environments.
- Triggering Code Manager with a webhook on page 798
  To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. The webhook is the simplest way to trigger Code Manager.
- Triggering Code Manager with custom scripts on page 799
Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repositories, or custom notifications.

**Understanding file sync and the staging directory**
To sync your code across your primary server and compilers, and to make sure that code stays consistent, Code Manager relies on file sync and two different code directories: the staging directory and the live code directory. Without Code Manager or file sync, Puppet code lives in the **codedir**, or live code directory, at `/etc/puppetlabs/code`.

With Code Manager and file sync, the file sync client service regularly checks for changes to staged code files in the staged code directory on the primary server at `/etc/puppetlabs/code-staging`. If it detects a change, the file sync client service fetches the changes and syncs the files to the live codedir on each compiler and the primary server (at `/etc/puppetlabs/code`). Because Code Manager moves new code from source control into the staging directory, and file sync moves it into the live code directory, you no longer write code in the codedir. If you manually edit the codedir, the next time Code Manager deploys code from source control, it overwrites your changes.

**Important:** Don’t directly modify code in the staging directory or live code directory (codedir). Code Manager overwrites the staging directory with changes from the control repo, and file sync overwrites the codedir with changes from the staging directory. Any changes made to these directories manually are overwritten.

**Related information**
- About file sync on page 828
  File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

**Environment isolation metadata and Code Manager**
The live code and staging code directories contain metadata files generated by file sync, which provide environment isolation for your resource types.

The metadata files, which have a `.pp` extension, ensure that each environment uses the correct version of the resource type.

**CAUTION:** Do not delete or modify the metadata files. Do not use expressions from these files in regular manifests.

These files are generated when Code Manager deploys new code in your environments. If you are new to Code Manager, these files are generated when you first deploy your environments. If you already use Code Manager, the files are generated as you make and deploy changes to your existing environments.

You can learn more about these files and their role in Environment isolation in the Puppet documentation.

**Moving from r10k to Code Manager**
Moving from r10k to Code Manager can improve automation of your code management and deployments.

While we recommend using Code Manager whenever possible, reasons you might not want to upgrade from r10k to Code Manager include:

- **Code Manager does not** allow you to manually deploy code with r10k. If you depend on the ability to deploy modules directly from r10k (with the `r10k deploy module` command), we recommend continuing to use r10k.
- **Code Manager** must deploy all control repositories to the same directory. If you use r10k to deploy control repositories to different directories, we recommend continuing to use r10k.
- **Code Manager** supports the shellgit provider, but only for HTTPS. It does not support system .SSH configuration or other shellgit options.
- **Code Manager** does not support post-deploy scripts.

If you rely on any of the above configurations Code Manager does not support, or if you are using a custom script to deploy code, carefully assess whether or not Code Manager can support your goals.
**Set up Code Manager**

You must set up Code Manager to use it as your code management tool.

To set up Code Manager, you must:

1. Prepare for Managing environments with a control repository on page 764. This involves creating a Git control repository that has a Puppetfile.

2. Create a control repository with Git for your code.
   
   Code Manager uses the control repo to maintain and deploy your Puppet code and data. You can also create separate deployment environments in your Puppet infrastructure by creating branches in your control repository (such as a development branch for a development environment). Code Manager tracks your environments and updates them according to the changes you make in your control repo.
   
   The Puppetfile specifies which modules and data to install in your environment, including what versions to install, and where to download the modules or other content.

3. Configure Code Manager on page 777.

4. Optional: Customize Code Manager configuration in Hiera on page 784

5. Deploy environments with a deployment trigger (recommended) or from the command line. For the initial configuration, you might prefer to use the command line, and then set up an automated trigger.
   
   - Triggering Code Manager on the command line on page 792
   - Triggering Code Manager with a webhook on page 798
   - Triggering Code Manager with custom scripts on page 799

**Configure Code Manager**

To configure Code Manager, first enable Code Manager in Puppet Enterprise (PE), then set up authentication, and test the communication between the control repository and Code Manager.

To configure Code Manager:

1. Create a control repo with a Puppetfile, as explained in Managing environments with a control repository on page 764.

2. Upgrade from r10k to Code Manager on page 778, if applicable.

3. Enable Code Manager on page 778.

4. Set up authentication for Code Manager on page 163.

5. Test the connection to the control repository on page 780.

6. Test Code Manager on page 780.
Depending on your needs, you might need to configure additional Code Manager settings on page 780, enable Lockless code deploys on page 781, or Customize Code Manager configuration in Hiera on page 784.

**Related information**

Set up Code Manager on page 777
You must set up Code Manager to use it as your code management tool.

Upgrade from r10k to Code Manager
To upgrade from r10k to Code Manager, you must disable the previous r10k installation.

1. Disable your previous r10k installation.
2. Disable any tools that automatically run r10k. Most commonly, it is the zack-r10k module.

   **Note:** When you upgrade to Code Manager, you can no longer manually use r10k or the zack-r10k module.

After you disable r10k, you can configure Code Manager.

**Related information**

Moving from r10k to Code Manager on page 776
Moving from r10k to Code Manager can improve automation of your code management and deployments.

Enable Code Manager
Enable Code Manager to connect your primary server to your Git repository.

**Before you begin**
You must have an SSH key without a passphrase and is saved on your primary server at /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.

1. In the console, in the PE Master node group, set parameters for the puppet_enterprise::profile::master class.
   - **code_manager_auto_configure** - Specify true to enable Code Manager.
   - **r10k_remote** - Enter a string that is a valid SSH URL for your Git control repository. For example: "git@<YOUR.GIT.SERVER.COM>:puppet/control.git".

   **Important:** Some Git providers have additional requirements for enabling SSH access. For example, BitBucket requires ssh:// at the beginning of the SSH URL (such as ssh:// git@<YOUR.GIT.SERVER.COM>:puppet/control.git). See your provider's documentation for this information.

   - **r10k_private_key** - Enter "/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519". This is the path to the private key that permits the pe-puppet user access to your Git repositories.
2. Run Puppet on your primary server and all compilers.

### Potential errors:

If you use **Run Puppet** in the console to trigger the Puppet run, the job, on the **Jobs** page, appears to fail due to underlying services being restarted. This error is not fatal and the **Reports** page shows the actual, successful result.

Additionally, if you run Puppet on your primary server and all compilers at the same time, the compilers’ logs might report these errors:

```
java.net.ConnectException: Connection refused
```

These errors occur when Puppet Server is restarting when the compilers poll for new code, and they usually stop when Puppet Server finishes restarting on the primary server. You can ignore these errors while the primary server starts.

### Set up authentication for Code Manager

To securely deploy environments, Code Manager needs an authentication token for both authentication and authorization.

To generate a token for Code Manager:

1. Assign a user to the deployment role.
2. In the console, create a deployment user.

   **Tip:** Create a dedicated deployment user for Code Manager use.

3. Add the deployment user to the **Code Deployers** role.

   **Note:** This role is automatically created on install, with default permissions for code deployment and token lifetime management.

4. Create a password by clicking **Generate Password**.
5. **Request an authentication token for deployments** on page 164

### Related information

**Configure puppet-access** on page 347

The **puppet-access** command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. Ensure puppet-access is configured correctly before using it to generate authentication tokens.

**Request an authentication token for deployments**

To enable secure deployment of your code, request an authentication token for the deployment user.

By default, authentication tokens have a one-hour lifetime. With the **Override default expiry** permission set, you can change the lifetime of the token to a duration better suited for a long-running, automated process.

1. From the command line on the primary server, run `puppet-access login --lifetime 180d`. This command both requests the token and sets the token lifetime to 180 days.

   **Tip:** You can add flags to the request to specify additional settings such as the token file's location or the URL for your RBAC API. See **Configuration file settings for puppet-access**.

2. Enter the user name and password of the deployment user when prompted.

The generated token is stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. To view the token, run `puppet-access show`. 

© 2022 Puppet, Inc., a Perforce company
**Related information**

Setting a token-specific lifetime on page 351
Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

Generate a token for use by a service on page 351
If you need to generate a token for use by a service and the token doesn't need to be saved, use the `--print` option.

**Test the connection to the control repository**
Verify if Code Manager connects to the control repository.

From the command line, run `puppet-code deploy --dry-run`.

- When the control repository is set up correctly, the command fetches and displays the number of environments in the control repository.
- Environments whose names contain invalid characters are excluded from the number of environments in the control repository. Only environments set up correctly are counted. Check the Puppet Server log for details about the errors.

**Test Code Manager**
Test Code Manager by deploying a single test environment.

Deploy a single test environment to test Code Manager:

From the command line, deploy one environment by running `puppet-code deploy my_test_environment --wait`.

This deploys the test environment, and then returns deployment results with the SHA (a checksum for the content stored) for the control repository commit.

If the environment deploys and returns the deployment results, Code Manager is correctly configured.

If the deployment does not work, review the configuration steps, or refer to Troubleshooting for help.

After Code Manager is fully enabled and configured, you can trigger Code Manager to deploy your environments.

There are several ways to trigger deployments, depending on your needs.

- Triggering Code Manager on the command line on page 792
- Triggering Code Manager with a webhook on page 798
- Triggering Code Manager with custom scripts on page 799

**Code Manager settings**
After Code Manager is configured, you can adjust its settings in the `PE Master` node group, in the `puppet Enterprise::profile::master` class.

These options are required for Code Manager to work, unless otherwise noted.

- `puppet Enterprise::profile::master::code_manager_auto_configure`
  Specifies whether to autoconfigure Code Manager and file sync.
  Default: `false`

- `puppet Enterprise::profile::master::r10k_remote`
  The location, as a valid URL, for your Git control repository.
  Example: `git@<YOUR.GIT.SERVER.COM>:puppet/control.git`

- `puppet Enterprise::profile::master::r10k_private_key`
  The path to the file containing the private key used to access all Git repositories. Required when using the SSH protocol; optional in all other cases.
  Example: `'/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519'`

© 2022 Puppet, Inc., a Perforce company
**puppet_enterprise::profile::master::r10k_proxy**

Optional proxy used by r10k when accessing the Forge. If empty, no proxy settings are used.

Example: 'http://proxy.example.com:3128'

**puppet_enterprise::profile::master::r10k_trace**

Configuration option that includes the r10k stacktrace in the error output of failed deployments when the value is true.

Default: false

**puppet_enterprise::profile::master::versioned_deploys**

Optional setting that specifies whether code is updated in versioned code directories instead of blocking requests and overwriting the live code directory.

Default: false

**Note:** For more information, see Lockless code deploys on page 781

**puppet_enterprise::master::environment_timeout**

Specifies whether and how long environments are cached, which can significantly reduce CPU usage of your Puppet Server. You can specify any of these values:

- 0 – no caching
- unlimited – all environments are cached forever
- a length of time for environments to be cached, for example 30m

Default when Code Manager is enabled: unlimited
Default when Code Manager is not enabled: 0

**puppet_enterprise::master::environment_timeout_mode**

Indicates whether a length of time specified by the environment_timeout begins when the environment is created (from_created) or when it's last used (from_last_used).

Default: from_created

**Note:** For optimal balance of performance and memory use, we recommend setting environment_timeout_mode to from_last_used and environment_timeout to 30m.

To further customize your Code Manager configuration with Hiera, see Customize Code Manager configuration in Hiera on page 784.

### Lockless code deploys

When deploying code, Puppet Server blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. However, you can enable lockless code deploys to deploy code without blocking requests to Puppet Server. With lockless code deploys, the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.

**Important:** Lockless code deploys require PE version 2019.8.7 or later.

With lockless deployments enabled, each new deploy writes code to versioned directories at:

```
/opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs/puppet-code/
```

When you enable lockless code deploys, the standard code directory (/etc/puppetlabs/code) is reconfigured to /etc/puppetlabs/puppetserver/code, which points, via symlink, to the most recent versioned code directory (at the versioned directories filepath specified above). If you disable lockless deploys after enabling it, your code directory moves back to the default location.
With lockless deploys, you can deploy a new version of code alongside an old version. When a catalog compiles starts, it uses the full path to the most recent version of code in the versioned code directory (via the /etc/puppetlabs/puppetserver/code symlink). Existing catalog compiles continue using the version they started on and new compiles use the latest code version.

To conserve disk space, code written to versioned directories is optimized to reduce duplication, and directories older than the latest and its predecessor are cleaned up after 30 minutes. If you deploy code very frequently, you might prefer to decrease the versioned-dirs-ttl setting, which is specified, in minutes, in file-sync.conf within each file sync client.

**Related information**

**Running plans alongside code deployments** on page 669
The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

**Enable lockless code deploys**
You can enable lockless code deploys on compilers individually or all at once. You must enable lockless code deploys on the primary server, the replica, and all compilers; however, we recommend testing one compiler before enabling lockless code deploys on all Puppet Server instances.

**Enable lockless code deploys on one compiler**
Use these steps to enable lockless code deploys on one Puppet Server instance.

1. In the compiler's node-specific Hiera file, set puppet-enterprise::profile::master::versioned_deploys to true.
2. Commit changes.
3. Run Puppet on the compiler twice: puppet agent -t; puppet agent -t
4. On your primary server, run: puppet code deploy --all --wait

   **Important:** You must deploy all environments (with --all) to avoid errors.

5. Allow some time (a few hours, a day, or longer) for the compiler to exist with lockless code deploys until you are satisfied with the result and ready to enable lockless code deploys elsewhere. The puppet_metrics_collector module, enabled by default, collects data about Puppet Server that you can use to monitor the impact on the compiler after enabling lockless code deploys.

If there are no errors and you're satisfied with the outcome, you're ready to enable lockless code deploys on all Puppet Server instances. You can either repeat these steps to enable lockless code deploys on all compilers, the primary server, and the replica individually, or Enable lockless code deploys on all Puppet Server instances on page 782 at once.

**Enable lockless code deploys on all Puppet Server instances**
Use enable lockless code deploys globally (on the primary server, the replica, and all compilers), you need to set the puppet-enterprise::profile::master::versioned_deploys parameter to true.

**Before you begin**
Enable lockless code deploys on one compiler on page 782 as a test before enabling lockless code deploys on all Puppet Server instances.

We recommend using Hiera to enable lockless code deploys globally, but you can also enable this setting in pe.conf or the PE console (as configuration data).

To use Hiera to enable lockless code deploys:
1. Open your default Hiera .yaml file in a text editor.

   **Tip:** For information about Hiera data files, including file paths, refer to Configure settings with Hiera on page 287.

2. Add the versioned_deploys parameter and set it to true. For example:

   puppet_enterprise::profile::master::versioned_deploys: true

3. Save the file and run Puppet on your primary server and all compilers twice:

   puppet agent -t; puppet agent -t

4. On your primary server, run: puppet code deploy --all --wait

   **Important:** You must deploy all environments (with --all) to avoid errors.

5. Commit changes.

6. Run Puppet on your primary server and all compilers twice: puppet agent -t; puppet agent -t

   **Tip:** You can also do this with a job. On the PE Master node group page, click Run > Puppet, select Override noop = true configuration, and click Run Job. When the job finishes, click Run again > All Nodes to run Puppet again.

   **Important:** You must deploy all environments (with --all) to avoid errors.

   You can use the --wait flag to return results after all deployments finish, but this might hit timeout limits in large installations. Alternately, you can check puppetserver.log to confirm that there were no errors during the code deployments.

**Related information**

Configure settings using the console on page 286

The console allows you to use a graphical interface to configure Puppet Enterprise (PE).

**System requirements for lockless deploys**

Enabling lockless deploys increases the disk storage required on your primary server and compilers because code is written to multiple versioned directories, instead of a single live code directory. Follow these guidelines for estimating your required system capacity.

You can roughly estimate your required disk storage with this equation:

\[
\text{(Size of typical environment)} \times \text{(Number of active environments)}
\]

For example, if your typical environment is 200 MB on disk when deployed, and you have 25 active environments, your disk storage calculation is 200 MB × 25, which equals 5,000 MB or 5 GB.

The number of times you deploy a given environment each day also impacts your disk use. Deploying multiple versions of the same environment uses approximately 25% more disk space than deploying multiple unique environments. To estimate the additional disk storage required for deploying environments multiple times a day, use this equation:

\[
\text{(Size of typical environment × .25)} \times \text{(Number of environments deployed multiple times per day)} \times \text{(Number of deployments per day)}
\]
Continuing the previous example, if 6 of your 200 MB environments are deployed up to 10 times per day, your additional disk storage calculation is \((200 \text{ MB} \times 0.25) \times 6 \times 10\), which equals 3,000 MB or 3 GB of additional disk space. In total, this example requires 8 GB available for your primary server and each compiler.

**Note:** If you're using the Continuous Delivery for PE impact analysis tool, you might need additional disk space beyond these estimates to accommodate the short-lived environments created during impact analysis.

### Customize Code Manager configuration in Hiera

To customize your Code Manager configuration, set parameters with Hiera.

**Important:** Do not manually edit the Code Manager configuration file. Puppet automatically manages this file and it overwrites or discards any manual changes you make.

To customize your configuration:

1. Add parameters to the `puppet_enterprise::master::code_manager` class in the `data/common.yaml` file in your control repo. Use the format:
   
   ```yaml
   puppet_enterprise::master::code_manager::<PARAMETER>: <SETTING>
   ```
   
   For example, these parameters increase the size of the default worker pool and reduce the maximum time allowed to deploy a single environment:
   
   ```yaml
   puppet_enterprise::master::code_manager::deploy_pool_size: 4
   puppet_enterprise::master::code_manager::timeouts_deploy: 300
   ```

   Refer to the other sections on this page for information about other settings you might want to configure.

2. Run Puppet on the primary server.

### Related information

- **Configure settings in pe.conf** on page 288
  Puppet Enterprise (PE) configuration data includes any data set in `/etc/puppetlabs/enterprise/conf.d/` but `pe.conf` is the file used for most configuration activities during installation.

- **Configure settings with Hiera** on page 287
  Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

- **Configuring post-environment hooks**
  Configure post-environment hooks to trigger custom actions after your environment deployment.

  To configure list of hooks to run after an environment has been deployed, specify the Code Manager `post_environment_hook` setting in Hiera.

  This parameter accepts an array of hashes, with the following keys:

  - `url`
  - `use-client-ssl`

  For example, to enable Code Manager to update classes in the console after deploying code to your environments.

  ```yaml
  puppet_enterprise::master::code_manager::post_environment_hooks:
     use-client-ssl: true
  ```
**url**

This setting specifies an HTTP URL to send a request to, with the result of the environment deploy. The URL receives a POST with a JSON body with the structure:

```json
{
    "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
    "file-sync": {
        "environment-commit": "6939889b679f679f6b679f649545c44f26aa06174d25c21",
        "code-commit": "ce5f715861579151f77391c7b2b8b497aebc1",
        "environment": "production",
        "id": 3,
        "status": "complete"
    }
}
```

**use-client-ssl**

Specifies whether the client SSL configuration is used for HTTPS connections. If the hook destination is a server using the Puppet CA, set this to `true`; otherwise, set it to `false`. Defaults to `false`.

**Configuring garbage collection**

By default, Code Manager retains environment deployments in memory for one hour, but you can adjust this by configuring garbage collection.

To configure the frequency of Code Manager garbage collection, specify the `deploy_ttl` parameter in Hiera. By default, deployments are kept for one hour.

Specify the time as a string using any of the following suffixes:

- d - days
- h - hours
- m - minutes
- s - seconds
- ms - milliseconds

For example, a value of `30d` would configure Code Manager to keep the deployment in memory for 30 days, and a value of `48h` would maintain the deployment in memory for 48 hours.

If the value of `deploy_ttl` is less than the combined values of `timeouts_fetch`, `timeouts_sync`, and `timeouts_deploy`, all completed deployments are retained indefinitely, which might significantly slow the performance of Code Manager over time.

**Configuring module deployment scope**

By default, Code Manager performs incremental deployments of module code. You can use the `full_deploy` parameter to change the module code deployment scope.

Incremental deploys only sync modules whose definitions (in the environment's Puppetfile) allow their version to "float" (such as Git branches) and modules whose definitions have been added or changed since the environment's last deployment. Incremental deploys do not support SVN modules.

If you want to deploy all module code regardless of change or float status, you can disable incremental deploys by setting the following parameter to `true`:

```erb
puppet_enterprise::master::code_manager::full_deploy
```

To re-enable incremental deploys, set the `full_deploy` parameter to `false`.

**Important:** There is a known issue impacting the ability to configure this parameter. A work around is described in [Code management known issues](https://docs.puppet.com/pe/managing/deploying.html#code-management-known-issues) on page 66.
Configuring Forge settings
To configure how Code Manager downloads modules from the Forge, specify `forge_settings` in Hiera.

This parameter configures where Forge modules are installed from, and sets a proxy for all Forge interactions. The `forge_settings` parameter accepts a hash with the following values:

- **baseurl**: Indicates where Forge modules are installed from. Defaults to `https://forgeapi.puppetlabs.com`.
- **authorization_token**: Specifies the token for authenticating to a custom Forge server.
- **proxy**: Sets the proxy for all Forge interactions.

**baseurl**
Indicates where Forge modules are installed from. Defaults to `https://forgeapi.puppetlabs.com`.

```
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: 'https://private-forge.mysite'
```

**authorization_token**
Specifies the token for authenticating to a custom Forge server. When set, `baseurl` must also be set, because the default Forge server does not require authentication. You must prepend the token with 'Bearer', particularly if you use Artifactory as your Forge server.

```
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: 'https://private-forge.mysite'
  authentication_token: 'Bearer mysupersecretauthtoken'
```

**proxy**
Sets the proxy for all Forge interactions. This setting overrides the global `proxy` setting on Forge operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. See the global `proxy` setting for more information and examples.

```
puppet_enterprise::master::code_manager::forge_settings:
  proxy: 'http://proxy.example.com:3128'
```

Configuring Git settings
To configure Code Manager to use a private key, a proxy, or multiple repositories with Git, specify the `git_settings` parameter.

The `git_settings` parameter accepts a hash of the following settings:

- **private-key**
- **proxy**
- **repositories**

**Note**: You cannot use the `git_settings` hash with the default Code Manager settings for `r10k_private_key`. To avoid errors, remove the `r10k_private_key` from the `puppet_enterprise::profile::master` class.

**private-key**
Specifies the file containing the default private key used to access control repositories, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`. This file must have read permissions for the `pe-puppet` user. The SSH key cannot require a password. This setting is required, but by default, the value is supplied in the `puppet_enterprise::profile::master` class.

© 2022 Puppet, Inc., a Perforce company
**proxy**

Sets a proxy specifically for Git operations that use an HTTP(S) transport.

This setting overrides the global proxy setting on Git operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. To set a proxy for a specific Git repository only, set proxy in the repositories subsetting of git_settings. See the global proxy setting for more information and examples.

```
proxy: 'http://proxy.example.com:3128'
```

**repositories**

Specifies a list of repositories and their respective private keys. Use this setting if you want to use multiple control repos.

To use multiple control repos, the sources setting and the repositories setting must match. Accepts the following settings:

- **remote**: The repository to which these settings apply.
- **private-key**: The file containing the private key to use for this repository. This file must have read permissions for the pe-puppet user.
- **proxy**: The proxy setting allows you to set or override the global proxy setting for a single, specific repository. See the global proxy setting for more information and examples.

The repositories setting accepts a hash in the following format:

```
repositories:
  - remote: "jfkennedy@gitserver.puppetlabs.net:repositories/repo_one.git"
    private-key: "/test_keys/jfkennedy"
  - remote: "whtaft@gitserver.puppetlabs.net:repositories/repo_two.git"
    private-key: "/test_keys/whtaft"
  - remote: "https://git.example.com/git_repos/environments.git"
    proxy: "https://proxy.example.com:3128"
```

**Configuring proxies**

To configure proxy servers, use the proxy setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

**proxy**

To set a proxy for all operations occurring over an HTTP(S) transport, set the global proxy setting. You can also set an authenticated proxy with either Basic or Digest authentication.

To override this setting for Git or Forge operations only, set the proxy subsetting under the git_settings or forge_settings parameters. To override for a specific Git repository, set a proxy in the repositories list of git_settings. To override this setting with no proxy for Git, Forge, or a particular repository, set that specific proxy setting to an empty string.

In this example, the first proxy is set up without authentication, and the second proxy uses authentication:

```
proxy: 'http://proxy.example.com:3128'
proxy: 'http://user:password@proxy.example.com:3128'
```

**Configuring sources**

If you are managing more than one repository with Code Manager, specify a map of your source repositories.

Use the sources parameter to specify a map of sources. Configure this setting if you are managing more than just Puppet environments, such as when you are also managing Hiera data in its own control repository.

To use multiple control repos, the sources setting and the repositories setting must match.
If `sources` is set, you cannot use the global `remote` setting. If you are using multiple sources, use the prefix option to prevent collisions.

The `sources` setting accepts a hash with the following subsettings:

- `remote`
- `prefix`

```ruby
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: "testing"
```

**remote**

Specifies the location from which you fetch the source repository. The remote must be able to be fetched without any interactive input. That is, you cannot be prompted for usernames or passwords in order to fetch the remote.

Accepts a string that is any valid URL that r10k can clone, such as `git://git-server.site/my-org/main-modules`.

**prefix**

Specifies a string to prefix to environment names. Alternatively, if `prefix` is set to `true`, the source's name is used. This prevents collisions when multiple sources are deployed into the same directory.

For example, with two "main-modules" environments set up in the same base directory, the prefix setting differentiates the environments: the first is named "myorg-main-modules", and the second is "testing-main-modules".

```ruby
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: "testing"
```

### Code Manager parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>remote</code></td>
<td>The location of the Git control repository. Either the <code>remote</code> or <code>sources</code> parameters must be specified. If <code>remote</code> is already specified in the <code>puppet_enterprise::profile::master</code> class, that value is used here. If both the <code>sources</code> and <code>remote</code> keys are specified, the <code>sources</code> key overrides <code>remote</code>.</td>
<td>A string that is a valid SSH URL for your Git remote.</td>
<td>No default.</td>
</tr>
<tr>
<td><code>authenticate_webhook</code></td>
<td>Turns on RBAC authentication for the /v1/webhook endpoint.</td>
<td>Boolean.</td>
<td>true</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>cachedir</td>
<td>The path to the location where Code Manager caches Git repositories.</td>
<td>A valid file path.</td>
<td>/opt/puppetlabs/server/data/code-manager/cache.</td>
</tr>
<tr>
<td>certname</td>
<td>The certname of the Puppet signed certs to use for SSL.</td>
<td>A valid certname.</td>
<td>$::clientcert.</td>
</tr>
<tr>
<td>data</td>
<td>The path to the directory where Code Manager stores internal file content.</td>
<td>A valid file path.</td>
<td>/opt/puppetlabs/server/data/code-manager</td>
</tr>
<tr>
<td>deploy_pool_size</td>
<td>Specifies the number of threads in the worker pool; this dictates how many deploy processes can run in parallel.</td>
<td>An integer.</td>
<td>2</td>
</tr>
<tr>
<td>download_pool_size</td>
<td>Specifies the number of threads used to download modules.</td>
<td>An integer.</td>
<td>4</td>
</tr>
<tr>
<td>deploy_ttl</td>
<td>Specifies the length of time completed deployments are retained before garbage collection. For usage details, see configuring garbage collection.</td>
<td>The time as a string, using any of the following suffixes: d - days, h - hours, m - minutes, s - seconds, ms - milliseconds</td>
<td>1h (one hour)</td>
</tr>
<tr>
<td>full_deploy</td>
<td>For Configuring module deployment scope on page 785.</td>
<td>Boolean.</td>
<td>false</td>
</tr>
<tr>
<td>hostcrl</td>
<td>The path to the SSL CRL.</td>
<td>A valid file path.</td>
<td>$puppet_enterprise::params::hostcrl</td>
</tr>
<tr>
<td>localcacert</td>
<td>The path to the SSL CA cert.</td>
<td>A valid file path.</td>
<td>$puppet_enterprise::params::localcacert</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>post_environment_hooks</td>
<td>A list of hooks to run after an environment has been deployed. Specifies:</td>
<td>An array of hashes that can include the keys:</td>
<td>No default.</td>
</tr>
<tr>
<td></td>
<td>• an HTTP URL to send deployment results to,</td>
<td>• url</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Whether to use client SSL for HTTPS connections</td>
<td>• use-client-ssl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For usage details, see configuring post-environment hooks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeouts_deploy</td>
<td>Maximum execution time (in seconds) for deploying a single environment.</td>
<td>An integer</td>
<td>600</td>
</tr>
<tr>
<td>timeouts_fetch</td>
<td>Maximum execution time (in seconds) for updating the control repo state.</td>
<td>An integer.</td>
<td>30</td>
</tr>
<tr>
<td>timeouts_hook</td>
<td>Maximum time (in seconds) to wait for a single post-environment hook URL to respond. Used for both the socket connect timeout and the read timeout, so the longest total timeout would be twice the specified value.</td>
<td>An integer.</td>
<td>30</td>
</tr>
<tr>
<td>timeouts_shutdown</td>
<td>Maximum time (in seconds) to wait for in-progress deploys to complete when shutting down the service.</td>
<td>An integer.</td>
<td>610</td>
</tr>
<tr>
<td>timeouts_wait</td>
<td>Maximum time that a request sent with a wait parameter should wait for each environment before timing out.</td>
<td>An integer.</td>
<td>700</td>
</tr>
<tr>
<td>timeouts_sync</td>
<td>Maximum time (in seconds) that a request sent with a wait parameter must wait for all compilers to receive deployed code before timing out.</td>
<td>An integer.</td>
<td>300</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>webserver_ssl_host</td>
<td>The host that Code Manager listens on.</td>
<td>An IP address</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>webserver_ssl_port</td>
<td>The port that Code Manager listens on. Port 8170 must be open if you're using Code Manager.</td>
<td>A port number.</td>
<td>8170</td>
</tr>
</tbody>
</table>

**r10k specific parameters**

These parameters are specific to r10k, which Code Manager uses in the background.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentdir</td>
<td>The single directory to which Code Manager deploys all sources. If file_sync_auto_commit is set to true, this defaults to /etc/puppetlabs/code-staging/environments. See file_sync_auto_commit.</td>
<td>Directory</td>
<td>/etc/puppetlabs/code-staging/environments</td>
</tr>
<tr>
<td>forge_settings</td>
<td>Contains settings for downloading modules from the Forge. See Configuring Forge settings for usage details.</td>
<td>Accepts a hash of:</td>
<td>No default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• baseurl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• proxy</td>
<td></td>
</tr>
<tr>
<td>invalid_branches</td>
<td>Specifies, for all sources, how branch names that cannot be cleanly mapped to Puppet environments are handled.</td>
<td>• 'correct': Replaces non-word characters with underscores and gives no warning.</td>
<td>'error'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 'error': Ignores branches with non-word characters and issues an error.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
</tbody>
</table>
| **git_settings** | Configures settings for Git:  
• Specifies the file containing the default private key used to access control repositories.  
• Sets or overrides the global proxy setting specifically for Git operations that use an HTTP(S) transport.  
• Specifies a list of repositories and their respective private keys.  
See Configuring Git settings for usage details. | Accepts a hash of:  
• private-key  
• proxy  
• repositories | Default private-key value as set in console. No other default settings. |
| **proxy**   | Configures a proxy server to use for all operations that occur over an HTTP(S) transport.  
See Configuring proxies for usage details. | Accepts:  
• A proxy server.  
• An authenticated proxy with Basic or Digest authentication.  
• An empty value. | No default. |
| **sources** | Specifies a map of sources to be passed to r10k.  
Use if you are managing more than just Puppet environments.  
See Configuring sources for usage details. | A hash of:  
• remote  
• prefix | No default. |

**Triggering Code Manager on the command line**

The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.

**Installing and configuring puppet-code**

PE automatically installs and configures the `puppet-code` command on your primary server as part of the included PE client tools package. You can also set up `puppet-code` on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.
The global configuration settings for Linux and macOS systems are in a JSON file at /etc/puppetlabs/client-tools/puppet-code.conf. The default configuration file looks something like:

```
{
  "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
  "token-file": "~/.puppetlabs/token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

**Important:** On a PE-managed machine, Puppet manages this file for you. Do not manually edit this file, because Puppet overwrites your new values the next time it runs.

Additionally, you can set up puppet-code on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files. For instructions, see the related topic about advanced configuration of puppet-code.

You can also override existing configuration settings on a case-by-case basis on the command line. When you deploy environments with the puppet-code deploy command, you can specify either an alternative config file or particular config settings directly in the command. For examples, see the related topic about deploying environments with puppet code.

**Windows paths**

The global puppet-code configuration file on Windows systems is located at C:\ProgramData\PuppetLabs\client-tools\puppet-code.conf.

The default configuration file looks something like:

```
{
  "cacert": "C:\\ProgramData\\PuppetLabs\\puppet\\etc\\ssl\\certs\\ca.pem",
  "token-file": "C:\\Users\\<username>\\.puppetlabs\\token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

**Configuration precedence and puppet-code**

There are several ways to configure puppet-code, but some configuration methods take precedence over others.

If no other configuration is specified, puppet-code uses the settings in the global configuration file. User-specific configuration files override the global configuration file.

If you specify a configuration file on the command line, Puppet temporarily uses that configuration file only and does not read the global or user-specific configuration files at all.

Finally, if you specify individual configuration options directly on the command line, those options temporarily take precedence over any configuration file settings.

**Deploying environments with puppet-code**

To deploy environments with the puppet-code command, use the deploy action, either with the name of a single environment or with the --all flag.

The deploy action deploys the environments, but returns only deployment queuing results by default. To view the results of the deployment itself, add the --wait flag.

The --wait flag deploys the specified environments, waits for file sync to complete code deployment to the live code directory and all compilers, and then returns results. In deployments that are geographically dispersed or have a large quantity of environments, completing code deployment can take up to several minutes.

The resulting message includes the deployment signature, which is the commit SHA of the control repo used to deploy the environment. The output also includes two other SHAs that indicate that file sync is aware that the environment has been newly deployed to the code staging directory.
To temporarily override default, global, and user-specific configuration settings, specify the following configuration options on the command line:

- `--cacert`
- `--token-file,-t`
- `--service-url`

Alternately, you can specify a custom `puppet-code.conf` configuration file by using the `--config-file` option.

**Running puppet-code on Windows**

If you're running these commands from a managed or non-managed Windows workstation, you must specify the full path to the command.

```
C:\Program Files\Puppet Labs\Client\bin\puppet code deploy mytestenvironment --wait
```

**Deploy environments on the command line**

To deploy environments, use the `puppet-code deploy` command, specifying the environments to deploy.

To deploy environments, on the command line, run `puppet-code deploy`, specifying the environment.

Specify the environment by name. To deploy all environments, use the `--all` flag.

Optionally, you can specify the `--wait` flag to return results after the deployment is finished. Without the `--wait` flag, the command returns only queuing results.

```
puppet-code deploy myenvironment --wait
puppet-code deploy --all --wait
```

Both of these commands deploy the specified environments, and then return deployment results with a control repo commit SHA for each environment.

**Related information**

*Reference: puppet-code command* on page 796

The `puppet-code` command accepts options, actions, and `deploy` action options.

**Deploy with a custom configuration file**

You can deploy environments with a custom configuration file that you specify on the command line.

To deploy all environments using the configuration settings in a specified config file, run the command `puppet-code deploy` command with a `--config-file` flag specifying the location of the config file.

```
puppet-code --config-file ~/.puppetlabs/myconfigfile/puppet code.conf deploy --all
```

**Deploy with command-line configuration settings**

You can override an existing configuration setting on a per-use basis by specifying that setting on the command line.

Specify the setting, which is used on this deployment only, on the command line.

```
puppet-code --service-url "https://puppet.example.com:8170/code-manager" deploy mytestenvironment
```

This example deploys 'mytestenvironment' using global or user-specific config settings (if set), except for `--service-url`, for which it uses the value specified on the command line ("https://puppet.example.com:8170/code-manager").
Advanced puppet-code configuration
You can set up the puppet-code command on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files.

You can set up the puppet-code command on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files.

The puppet-code.conf file is a JSON configuration file for the puppet-code command. For Linux or Mac OS X operating systems, it looks something like:

```
{
    "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
    "token-file": "~/.puppetlabs/token",
    "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

For Windows systems, use the entire Windows path, such as:

```
{
    "cacert": "C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem",
    "token-file": "C:\Users\<username>\.puppetlabs\token",
    "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

Related information
Installing client tools on page 250
PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

Configure puppet-code on agents and workstations
To use puppet-code on an agent node or on a workstation that is not managed by PE, install the client tools package and configure puppet-code on that machine.

Before you begin
Download and install the client tools package. See the client tools documentation for instructions.

Create a config file called puppet-code.conf in the client tools directory.

- For Linux and Mac OS X systems, the default client tools directory is /etc/puppetlabs/client-tools
- For Windows systems, the default client tools directory is C:\ProgramData\PuppetLabs\client-tools

Configure puppet-code for different users
On any machine, whether it is a primary server, an agent, or a workstation not managed by PE, you can set up specific puppet-code configurations for specific users.

Before you begin
If PE is not installed on the workstation you are configuring, see instructions for configuring puppet-code on agents and workstations.

1. Create a puppet-code.conf file in the user's client tools directory.
   - For Linux or Mac OS X systems, place the file in the user's ~/.puppetlabs/client-tools/
   - For Windows systems, place the file in the default user config file location: C:\Users\<username>\puppetlabs\ssl\certs\ca.pem

2. In the user's puppet-code.conf file, specify the cacert, token-file, and service-url settings in JSON format.
Reference: puppet-code command
The puppet-code command accepts options, actions, and deploy action options.

Usage: puppet-code [global options] <action> [action options]

Global puppet-code options
Global options set the behavior of the puppet-code command on the command line.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Accepted arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help, -h</td>
<td>Prints usage information for puppet-code.</td>
<td>none</td>
</tr>
<tr>
<td>--version, -V</td>
<td>Prints the application version.</td>
<td>none</td>
</tr>
<tr>
<td>--log-level, -l</td>
<td>Sets the log verbosity. It accepts one log level as an argument.</td>
<td>log levels: none, trace, debug, info, warn, error, fatal.</td>
</tr>
<tr>
<td>--config-file, -c</td>
<td>Sets a puppet-code.conf file that takes precedence over all other existing puppet-code.conf files.</td>
<td>A path to a puppet-code.conf file.</td>
</tr>
<tr>
<td>--cacert</td>
<td>Sets a Puppet CA certificate that overrides the cacert setting in any configuration files.</td>
<td>A path to the location of a CA Certificate.</td>
</tr>
<tr>
<td>--token-file, -t</td>
<td>Sets an authentication token that overrides the token-file setting in any configuration files.</td>
<td>A path to the location of the authentication token.</td>
</tr>
<tr>
<td>--service-url</td>
<td>Sets a base URL for the Code Manager service, overriding service-url settings in any configuration files.</td>
<td>A valid URL to the service.</td>
</tr>
</tbody>
</table>

puppet-code actions
The puppet-code command can perform print, deploy, and status actions.

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
<th>Arguments</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet-code print-config</td>
<td>Prints out the resolved puppet-code configuration.</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>puppet-code deploy</td>
<td>Runs remote code deployments with the Code Manager service. By default, returns only deployment queuing results. Accepts either the name of a single environment or the --all flag.</td>
<td>Accepts either the name of a single environment or the --all flag.</td>
<td>--wait, -w; --all; --dry-run; --format, -F</td>
</tr>
<tr>
<td>puppet-code status</td>
<td>Checks whether Code Manager and file sync are responding. By default, details are returned at the info level. Accepts log levels none, trace, info, warn, error, fatal.</td>
<td>Accepts log levels none, trace, info, warn, error, fatal.</td>
<td>none</td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
**puppet-code deploy action options**
Modify the `puppet-code deploy` action with action options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `--wait, -w`| Causes `puppet-code deploy` to:  
  1. Start a deployment,  
  2. Wait for the deployment to complete,  
  3. Wait for file sync to deploy the code to all compilers, and  
  4. Return the deployment signature with control repository commit SHAs for each environment.  
  The return output also includes two other SHAs that indicate that file sync is aware that the environment has been newly deployed to the code-staging directory. |
| `--all`     | Tells `puppet-code deploy` to start deployments for all Code Manager environments.                                                                                                                             |
| `--dry-run` | Tests the connections to each configured remote and, if successfully connected, returns a consolidated list of the environments from all remotes. The `--dry-run` flag implies both `--all` and `--wait`. |}

**puppet-code.conf configuration settings**
Temporarily specify `puppet-code.conf` configuration settings on the command line.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Controls</th>
<th>Linux and Mac OS X default</th>
<th>Windows default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cacert</code></td>
<td>Specifies the path to the Puppet CA certificate to use when connecting to the Code Manager service over SSL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>token-file</code></td>
<td>Specifies the location of the file containing the authentication token for Code Manager.</td>
<td>/etc/puppetlabs/token</td>
<td>C:\Users&lt;username&gt;.puppetlabs\token</td>
</tr>
<tr>
<td><code>service-url</code></td>
<td>Specifies the base URL of the Code Manager service</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2022 Puppet, Inc., a Perforce company
Triggering Code Manager with a webhook

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. The webhook is the simplest way to trigger Code Manager.

Custom scripts are a good alternative if you have requirements such as existing continuous integration systems (including Continuous Delivery for Puppet Enterprise (PE)), privately hosted Git repos, or custom notifications. For information about writing a script to trigger Code Manager, see the related topic about creating custom scripts.

Code Manager supports webhooks for GitHub, Bitbucket Server (formerly Stash), Bitbucket, and Team Foundation Server. The webhook must only be used by the control repository. It can't be used by any other repository (for example, other internal component module repositories).

**Important:** Code Manager webhooks are not compatible with Continuous Delivery for PE. If your organization uses Continuous Delivery for PE, you must use a method other than webhooks to deploy environments.

Creating a Code Manager webhook

To set up the webhook to trigger environment deployments, you must create a custom URL, and then set up the webhook with your Git host.

**Creating a custom URL for the Code Manager webhook**

To trigger deployments with a webhook, you’ll need a custom URL to enable communication between your Git host and Code Manager.

Code Manager supports webhooks for GitHub, Bitbucket Server (formerly Stash), Bitbucket, GitLab (Push events only), and Team Foundation Server (TFS). To use the GitHub webhook with the Puppet signed cert, disable SSL verification.

To create the custom URL for your webhook, use the following elements:

- The name of the primary server (for example, `code-manager.example.com`).
- The Code Manager port (for example, 8170).
- The endpoint (`/code-manager/v1/webhook/`).
- Any relevant query parameters (for example, `type=github`).
- Your authentication token for deployments (`token=<TOKEN>`), passed with the `token` query parameter. To generate a token, see [Request an authentication token for deployments](#) on page 164.

For example, the URL for a GitHub webhook might look like this:

```plaintext
https://code-manager.example.com:8170/code-manager/v1/webhook?type=github&token=<TOKEN>
```

The URL for a version 5.4 or later Bitbucket Server webhook might look something like this:

```plaintext
```

With the complete token attached, a GitHub URL looks something like this:

```plaintext
https://code-manager.example.com:8170/code-manager/v1/webhook?type=github&token=0WJ4YPJvYQz26xm3X2I0ihb7MUa6812CZWjxM3vt4mQ
```

**Code Manager webhook query parameters**

The following query parameters are permitted in the Code Manager webhook.

The `token` query is mandatory, unless you disable `authenticate_webhook` in the Code Manager configuration.
• **type**: Required. Specifies which type of post body to expect. Accepts:
  - `github`: GitHub
  - `gitlab`: GitLab
  - `bitbucket-server`: Bitbucket Server version 5.4 or later (formerly Stash)
  - `bitbucket`: Bitbucket
  - `tfs-git`: Team Foundation Server (resource version 1.0 is supported)

• **prefix**: Specifies a prefix for converting branch names to environments.

  **Important**: You must provide the `prefix` parameter if the configuration of your control repository uses prefixing. If you do not specify this parameter, Code Manager cannot locate and deploy the correct environment or translate a branch name to the applicable environment name. See [Configuring Sources](#) in the related links for more information.

• **token**: Specifies the entire PE authorization token.

**Related information**

[Configuring sources](#) on page 787

If you are managing more than one repository with Code Manager, specify a map of your source repositories.

**Setting up the Code Manager webhook on your Git host**

Enter the custom URL you created for Code Manager into your Git server’s webhook form as the payload URL.

The content type for webhooks is JSON.

Exactly how you set up your webhook varies, depending on where your Git repos are hosted. For example, in your GitHub repo, click on **Settings > Webhooks & services** to enter the payload URL and enter `application/json` as the content type.

**Tip**: On Bitbucket Server, the server configuration menu has settings for both "hooks" and "webhooks." To set up Code Manager, use the webhooks configuration. For proper webhook function with Bitbucket Server, make sure you are using the Bitbucket Server 5.4 or later and the latest fix version of PE.

After you've set up your webhook, your Code Manager setup is complete. When you commit new code and push it to your control repo, the webhook triggers Code Manager, and your code is deployed.

**Testing and troubleshooting a Code Manager webhook**

To test and troubleshoot your webhook, review your Git host logs or check the Code Manager troubleshooting guide.

Each of the major repository hosting services (such as GitHub or Bitbucket) provides a way to review the logs for your webhook runs, so check their documentation for instructions.

For other issues, check the Code Manager troubleshooting for some common problems and troubleshooting tips.

**Triggering Code Manager with custom scripts**

Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repositories, or custom notifications.

Alternatively, a webhook provides a simpler way to trigger deployments automatically.

To create a script that triggers Code Manager to deploy your environments, you can use either the `puppet-code` command or a `curl` statement that hits the endpoints. We recommend the `puppet-code` command.

Either way, after you have composed your script, you can trigger deployment of new code into your environments. Commit new code, push it to your control repo, and run your script to trigger Code Manager to deploy your code.

All of these instructions assume that you have enabled and configured Code Manager.

**Related information**

[Request an authentication token for deployments](#) on page 164
To enable secure deployment of your code, request an authentication token for the deployment user.

**Code Manager API** on page 804
Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

**Deploying environments with the puppet-code command**

The `puppet-code` command allows you to trigger environment deployments from the command line. You can use this command in your custom scripts.

For example, to deploy all environments and then return deployment results with commit SHAs for each of your environments, incorporate this command into your script:

```
puppet-code deploy --all --wait
```

**Related information**

**Triggering Code Manager on the command line** on page 792
The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.

**Deploying environments with a curl command**

To trigger Code Manager code deployments with a custom script, compose a `curl` command to hit the Code Manager /deploys endpoint.

To deploy environments with a `curl` command in your custom script, compose a `curl` command to hit the `/v1/deploys` endpoint.

Specify either the "deploy-all" key, to deploy all configured environments, or the "environments" key, to deploy a specified list of environments. By default, Code Manager returns queuing results immediately after accepting the deployment request.

If you prefer to get complete deployment results after Code Manager finishes processing the deployments, specify the optional "wait" key with your request. In deployments that are geographically dispersed or have a large number of environments, completing code deployment can take up to several minutes.

For complete information about Code Manager endpoints, request formats, and responses, see the Code Manager API documentation.

You must include a custom URL for Code Manager and a PE authentication token in any request to a Code Manager endpoint.

**Creating a custom URL for Code Manager scripts**

To trigger deployments with a `curl` command in a custom script, you'll need a custom Code Manager URL. This URL is composed of:

- Your primary server name (for example, `primary.example.com`)
- The Code Manager port (for example, 8170)
- The endpoint you want to hit (for example, `/code-manager/v1/deploys/`)
- The authentication token you generated earlier (`token=<TOKEN>`)

A typical URL for use with a `curl` command might look something like this:

```
https://primary.example.com:8170/code-manager/v1/deploys&token=<TOKEN>
```

**Attaching an authentication token**

You must attach a PE authentication token to any request to Code Manager endpoints. You can either:
• Specify the path to the token with `puppet-access show` in the body of your request. For example:

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{"environments": ["production"], "wait": true}'
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

• Attach the entire token to your Code Manager URL using the `token` query parameter. For example:

```bash
https://$(puppet config print server):8170/code-manager/v1/webhook?type=github&token=<TOKEN>
```

If you are using a curl command to hit the `/webhook` endpoint directly, you must attach the entire token.

**Checking deployment status with a curl command**

You can check the status of a deployment by hitting the `status` endpoint.

To use a curl command in your custom script, compose a curl command to hit the status endpoint. You must incorporate a custom URL for Code Manager in the script.

**Troubleshooting Code Manager**

Code Manager requires coordination between multiple components, including r10k and the file sync service. If you have issues with Code Manager, check that these components are functioning.

**Code Manager logs**

Code Manager logs to the Puppet Server log. By default, this log is in `/var/log/puppetlabs/puppetserver/puppetserver.log`. For more information about working with the logs, see the Puppet Server logs documentation.

**Check Code Manager status**

Check the status of Code Manager and file sync if your deployments are not working as expected, or if you need to verify that Code Manager is enabled before running a dependent command.

The command `puppet-code status` verifies that Code Manager and file sync are responding. The command returns the same information as the Code Manager status endpoint. By default, the command returns details at the info level; critical and debug aren’t supported.

The following table shows errors that might appear in the `puppet-code status` output.

<table>
<thead>
<tr>
<th>Error</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Manager couldn’t connect to the server</td>
<td><code>pe-puppetserver</code> process isn’t running</td>
</tr>
<tr>
<td>Code Manager reports invalid configuration</td>
<td>Invalid configuration at <code>/etc/puppetlabs/puppetserver/conf.d/code-manager.conf</code></td>
</tr>
<tr>
<td>File sync storage service reports unknown status</td>
<td>Status callback timed out</td>
</tr>
</tbody>
</table>

**Test the connection to the control repository**

The control repository controls the creation of environments, and ensures that the correct versions of all the necessary modules are installed. The primary server must be able to access and clone the control repo as the `pe-puppet` user.
To make sure that Code Manager can connect to the control repo, run:

```
puppet-code deploy --dry-run
```

If the connection is set up correctly, this command returns a list of all environments in the control repo or repos. If the command completes successfully, the SSH key has the correct permissions, the Git URL for the repository is correct, and the pe-puppet user can perform the operations involved.

If the connection is not working as expected, Code Manager reports an `Unable to determine current branches for Git source` error.

The unsuccessful command also returns a path on the primary server that you can use for debugging the SSH key and Git URL.

**Check the Puppetfile for errors**

Check the Puppetfile for syntax errors and verify that every module in the Puppetfile can be installed from the listed source. To do this, you need a copy of the Puppetfile in a temporary directory.

Create a temporary directory `/var/tmp/test-puppetfile` on the primary server for testing purposes, and place a copy of the Puppetfile into the temporary directory.

You can then check the syntax and listed sources in your Puppetfile.

**Check Puppetfile syntax**

To check the Puppetfile syntax, run `r10k puppetfile check` from within the temporary directory.

If you have Puppetfile syntax errors, correct the syntax and test again. When the syntax is correct, the command prints "Syntax OK".

**Check the sources listed in the Puppetfile**

To test the configuration of all sources listed in your Puppetfile, perform a test installation. This test installs the modules listed in your Puppetfile into a modules directory in the temporary directory.

In the temporary directory, run the following command:

```
sudo -H -u pe-puppet bash -c \\
'/opt/puppetlabs/puppet/bin/r10k puppetfile install'
```

This installs all modules listed in your Puppetfile, verifying that you can access all listed sources. Take note of all errors that occur. Issues with individual modules can cause issues for the entire environment. Errors with individual modules (such as Git URL syntax or version issues) show up as general errors for that module.

If you have modules from private Git repositories requiring an SSH key to clone the module, check that you are using the SSH Git URL and not the HTTPS Git URL.

After you've fixed errors, test again and fix any further errors, until all errors are fixed.

**Run a deployment test**

Manually run a full r10k deployment to check not only the Puppetfile syntax and listed host access, but also whether the deployment works through r10k.

This command attempts a full r10k deployment based on the `r10k.yaml` file that Code Manager uses. This test writes to the code staging directory only. This does not trigger a file sync and is to be used only for ad-hoc testing.

Run this deployment test with the following command:

```
sudo -H -u pe-puppet bash -c \\
```

© 2022 Puppet, Inc., a Perforce company
'opt/puppetlabs/puppet/bin/r10k deploy environment -c /opt/puppetlabs/server/data/code-manager/r10k.yaml -p -v debug'

If this command completes successfully, the /etc/puppetlabs/code-staging directory is populated with directory-based environments and all of the necessary modules for every environment.

If the command fails, the error is likely in the Code Manager settings specific to r10k. The error messages indicate which settings are failing.

**Monitor logs for webhook deployment trigger issues**

Issues that occur when a Code Manager deployment is triggered by the webhook can be tricky to isolate. Monitor the logs for the deployment trigger to find the issue.

Deployments triggered by a webhook in Stash/Bitbucket, GitLab, or GitHub are governed by authentication and hit each service's /v1/webhook endpoint.

If you are using a GitLab version older than 8.5.0, Code Manager webhook authentication does not work because of the length of the authentication token. To use the webhook with GitLab, either disable authentication or update GitLab.

**Note:** If you disable webhook authentication, it is disabled only for the /v1/webhook endpoint. Other endpoints (such as /v1/deploys) are still controlled by authentication. There is no way to disable authentication on any other Code Manager endpoint.

To troubleshoot webhook issues, follow the Code Manager webhook instructions while monitoring the Puppet Server log for successes and errors. To do this, open a terminal window and run:

```
tail -f /var/log/puppetlabs/puppetserver/puppetserver.log
```

Watch the log closely for errors and information messages when you trigger the deployment. The puppetserver.log file is the only location these errors appear.

If you cannot determine the problem with your webhook in this step, manually deploy to the /v1/deploys endpoint, as described in the next section.

**Monitor logs for /v1/deploys deployment trigger issues**

Issues that occur when a Code Manager deployment is triggered by the /v1/deploys endpoint can be tricky to isolate. Monitor the logs for the deployment trigger to find the issue.

Before you trigger a deployment to the /v1/deploys endpoint, generate an authentication token. Then deploy one or more environments with the following command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{"environments": ["production"], "wait": true}'
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$url" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

This request waits for the deployment and sync to compilers to complete ("wait": true) and so returns errors from the deployment.

Alternatively, you can deploy all environments with the following curl command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
url="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{"deploy-all": true, "wait": true}'
```
curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"

Monitor the console-services.log file for any errors that arise from this curl command.

tail -f /var/log/puppetlabs/console-services/console-services.log

**Code deployments time out**
If your environments are heavily loaded, code deployments can take a long time, and the system can time out before deployment is complete.

If your deployments are timing out too soon, increase one or more of these settings:

- timeouts_deploy
- timeouts_shutdown
- timeouts_sync
- timeouts_wait

For descriptions of these settings, refer to Code Manager parameters on page 788. For instructions on configuring these settings, refer to Customize Code Manager configuration in Hiera on page 784.

Tip: Deployment timeouts can also occur when the file sync client holds code deployments while waiting for long-running plans to finish. To resolve this, you can increase the timeouts_sync setting or allow Running plans alongside code deployments on page 669.

**File sync stops when Code Manager tries to deploy code**
Code Manager code deployment involves accessing many small files. If you store your Puppet code on network-attached storage, slow network or backend hardware can result in poor deployment performance.

Tune the network for many small files, or store Puppet code on local or direct-attached storage.

**Classes are missing after deployment**
After a successful code deployment, a class you added isn't showing in the console.

If your code deployment works, but a class you added isn't in the console:

1. Check on disk to verify that the environment folder exists.
2. Check your node group in the Edit node group metadata box to make sure it's using the correct environment.
3. Refresh classes.
4. Refresh your browser.

**Code Manager API**
Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

**Authentication**
All Code Manager endpoint requests require token-based authentication. For /deploys endpoints, you can pass this token as either an HTTP header or as a query parameter. For the /webhook endpoint, you must attach the entire token with the token query parameter.

To generate an authentication token, from the command line, run puppet-access login. This command stores the token in ~/.puppetlabs/token. To learn more about authentication for Code Manager use, see the related topic about requesting an authentication token.

Pass this token as either an X-Authentication HTTP header or as the token query parameter with your request. If you pass the token as a query parameter, it might show up in access logs.
For example, an HTTP header for the /deploys endpoint looks something like:

```plaintext
type_header='Content-Type: application/json'
auth_header='X-Authentication: $(puppet-access show)'
uri=https://$(puppet config print server):8170/code-manager/v1/deploys

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See Usage notes for curl examples for information about forming curl commands.

```plaintext
$pe_headers = @{
  "X-Authentication" = "AMi1Wlel2Ybd2Lqu-hdhHaLWCkfvgsGl8AUgKdwzixwe" ;
}


$pe_codemgr_hash = [ordered]@{
  environments = @("production")
}

# Generate JSON object from pe_codemgr_hash
$JSON = $pe_codemgr_hash | convertto-json -compress
Invoke-RestMethod -SkipCertificateCheck -ContentType "application/json" -uri $pe_uri -Method POST -Body $JSON -Headers $pe_headers
```

Passing the token with the token query parameter might look like:

```plaintext
type_header='Content-Type: application/json'
auth_header='X-Authentication: $(puppet-access show)'
uri=https://$(puppet config print server):8170/code-manager/v1/webhook?type=github&token=$(puppet-access show)

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

# Common Data
$token = "AMi1Wlel2Ybd2Lqu-hdhHaLWCkfvgsGl8AUgKdwzixwe"


# Example 1 - deploys

$pe_headers = @{
  "X-Authentication" = "$token" ;
}

$pe_codemgr_hash = [ordered]@{
  environments = @("production")
}

# Generate JSON object from pe_codemgr_hash
$JSON = $pe_codemgr_hash | convertto-json -compress
Invoke-RestMethod -SkipCertificateCheck -ContentType "application/json" -uri $pe_uri/v1/deploys" -Method POST -Body $JSON -Headers $pe_headers

# Example 2 - Web hook

```bash
$pe_ex2_uri = "$pe_uri/v1/webhook?token=$token&type=github"
Write-Output "PE Example 2 URI: $pe_ex2_uri"
Invoke-RestMethod -SkipCertificateCheck -uri $pe_ex2_uri -Method POST -
ContentType "application/json"
```

**Related information**

**API index** on page 32
APIs allow you to interact with Puppet and Puppet Enterprise (PE) applications from your own code or application integration hooks.

**Request an authentication token for deployments** on page 164
To enable secure deployment of your code, request an authentication token for the deployment user.

**POST /v1/deploys**
Use the Code Manager/deploys endpoint to queue code deployments.

**Request format**
The body of the POST /deploys request must be a JSON object that includes the authentication token and a URL containing:

- The name of the primary server, such as `primary.example.com`
- The Code Manager port, such as 8170.
- The endpoint.

The request must also include either:

- The "environments" key, with or without the "wait" key.
- The "deploy-all" key, with or without the "wait" key.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;environments&quot;</td>
<td>Specifies the environments for which to queue code deployments. If not specified, the &quot;deploy-all&quot; key must be specified.</td>
<td>Accepts an array of strings specifying environments.</td>
</tr>
<tr>
<td>&quot;deploy-all&quot;</td>
<td>Specifies whether Code Manager deploys all environments or not.</td>
<td><code>true,false</code></td>
</tr>
</tbody>
</table>

If specified `true`, Code Manager determines the most recent list of environments and queues a deploy for each one. If specified `false` or not specified, the "environments" key must be specified with a list of environments.
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;deploy-modules&quot;</td>
<td>Specifies whether Code Manager deploys modules from the Puppetfile.</td>
<td>true, false</td>
</tr>
<tr>
<td></td>
<td>If specified true or not specified, Code Manager deploys modules. If</td>
<td></td>
</tr>
<tr>
<td></td>
<td>specified false, modules aren’t deployed.</td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>To ensure environments are populated before use, modules are deployed on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the first deployment of an environment, even if you pass &quot;deploy-modules&quot;:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>false.</td>
<td></td>
</tr>
<tr>
<td>&quot;modules&quot;</td>
<td>Specifies the modules to be deployed to the selected environments.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accepts a comma- and space-separated list of modules to deploy.</td>
<td></td>
</tr>
<tr>
<td>&quot;wait&quot;</td>
<td>Specifies whether Code Manager must wait for all deploys to finish or</td>
<td>true, false</td>
</tr>
<tr>
<td></td>
<td>error before it returns a response.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If specified true, Code Manager returns a response after all deploys have</td>
<td></td>
</tr>
<tr>
<td></td>
<td>either finished or errored. If specified false or not specified, returns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a response showing that deploys are queued.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify together with the &quot;deploy-all&quot; or &quot;environment&quot; keys.</td>
<td></td>
</tr>
<tr>
<td>&quot;dry-run&quot;</td>
<td>Tests the Code Manager connection to each of the configured remotes and</td>
<td>true, false</td>
</tr>
<tr>
<td></td>
<td>attempts to fetch a list of environments from each, reporting any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>connection errors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify together with the &quot;deploy-all&quot; or &quot;environment&quot; keys.</td>
<td></td>
</tr>
</tbody>
</table>

For example, to deploy the production and testing environments and return results after the deployments complete or fail, pass the "environments" key with the "wait" key:

```plaintext
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
```
data='{"environments": ["production", "testing"], "wait": true}'
curl --header "$type_header" --header "$auth_header" --cacert "$cacert" --request POST "$uri" --data "$data"

See Usage notes for curl examples for information about forming curl commands.

To deploy all configured environments and return only the queuing results, pass the "deploy-all" key without the "wait" key:

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{"deploy-all": true}'
curl --header "$type_header" --header "$auth_header" --cacert "$cacert" --request POST "$uri" --data "$data"

Response format

The POST /deploys response contains a list of objects, each containing data about a queued or deployed environment.

If the request did not include a "wait" key, the response returns each environment's name and status.

If the "wait" key was included in the request, the response returns information about each deployment. This information includes a deploy-signature, which is the commit SHA of the control repo that Code Manager used to deploy the environment. The output also includes two other SHAs that indicate that file sync is aware that the environment has been deployed to the code staging directory.

The response shows any deployment failures, even after they have successfully deployed. The failures remain in the response until cleaned up by garbage collection.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;environment&quot;</td>
<td>The name of the environment queued or deployed.</td>
<td>A string specifying the name of an environment.</td>
</tr>
<tr>
<td>&quot;id&quot;</td>
<td>Identifies the queue order of the code deploy request.</td>
<td>An integer generated by Code Manager.</td>
</tr>
<tr>
<td>&quot;status&quot;</td>
<td>The status of the code deployment for that environment.</td>
<td>Can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;new&quot;: The deploy request has been accepted but not yet queued.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;complete&quot;: The deploy is complete and has been synced to the live code directory on primary server and compilers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;failed&quot;: The deploy failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;queued&quot;: The deploy is queued and waiting.</td>
</tr>
<tr>
<td>&quot;deploy-signature&quot;</td>
<td>The commit SHA of the control repo that Code Manager used to deploy code in that environment.</td>
<td>A Git SHA.</td>
</tr>
</tbody>
</table>
Managing and deploying Puppet code

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;file-sync&quot;</td>
<td>Commit SHAs used internally by file sync to identify the code synced to the code staging directory</td>
<td>Git SHAs for the &quot;environment-commit&quot; and &quot;code-commit&quot; keys.</td>
</tr>
</tbody>
</table>

For example, for a /deploys request without the "wait" key, the response shows only "new" or "queued" status, because this response is returned as soon as Code Manager accepts the request.

```
[
  {
    "environment": "production",
    "id": 1,
    "status": "queued"
  },
  {
    "environment": "testing",
    "id": 2,
    "status": "queued"
  }
]
```

If you pass the "wait" key with your request, Code Manager doesn't return any response until the environment deployments have either failed or completed. For example:

```
[
  {
    "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
    "file-sync": {
      "environment-commit": "6939889b679fdb1449545c44f26aa06174d25c21",
      "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebc1"},
    "environment": "production",
    "id": 3,
    "status": "complete"
  }
]
```

Error responses

If any deployments fail, the response includes an error object for each failed environment. Code Manager returns deployment results only if you passed the "wait" key; otherwise, it returns queue information.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;environment&quot;</td>
<td>The name of the environment queued or deployed.</td>
<td>A string specifying the name of an environment.</td>
</tr>
<tr>
<td>&quot;error&quot;</td>
<td>Information about the deployment failure.</td>
<td>Contains keys with error details.</td>
</tr>
<tr>
<td>&quot;corrected-name&quot;</td>
<td>The name of the environment.</td>
<td>A string specifying the name.</td>
</tr>
<tr>
<td>&quot;kind&quot;</td>
<td>The kind of error encountered.</td>
<td>An error type.</td>
</tr>
<tr>
<td>&quot;msg&quot;</td>
<td>The error message.</td>
<td>An error message.</td>
</tr>
<tr>
<td>&quot;id&quot;</td>
<td>Identifies the queue order of the code deploy request.</td>
<td>An integer generated by Code Manager.</td>
</tr>
<tr>
<td>&quot;status&quot;</td>
<td>The status of the requested code deployment.</td>
<td>For errors, status is &quot;failed&quot;.</td>
</tr>
</tbody>
</table>
For example, information for a failed deploy of an environment might look like:

```json
{
    "environment": "test14",
    "error": {
        "details": {
            "corrected-name": "test14"
        },
        "kind": "puppetlabs.code-manager/deploy-failure",
        "msg": "Errors while deploying environment 'test14' (exit code: 1):
ERROR\t -> Authentication failed for Git remote "https://github.com/puppetlabs/puffpetlabs-apache".\n",
        "id": 52,
        "status": "failed"
    }
}
```

**POST /v1/webhook**

Use the Code Manager /webhook endpoint to trigger code deploys based on your control repo updates.

**Request format**

The POST /webhook request consists of a specially formatted URL that specifies the webhook type, an optional branch prefix, and a PE authentication token.

You must pass the authentication token with the token query parameter. To use a GitHub webhook with the Puppet signed certificate, you must disable SSL verification.

In your URL, include:

- The name of the primary server (for example, primary.example.com)
- The Code Manager port (for example, 8170)
- The endpoint (/v1/webhook)
- The type parameter with a valid value, such as gitlab.
- The prefix parameter and value.
- The token parameter with the complete token.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
</table>
| type      | Required. Specifies what type of POST body to expect.           | - github
|           |                                                                  |   - gitlab
|           |                                                                  |   - tfs-git
|           |                                                                  |   - bitbucket
|           |                                                                  |   - stash
| prefix    | Optional. Specifies a prefix to use in translating branch names to environments. | A string specifying a branch prefix. |
| token     | Specifies the PE authorization token to use. Required unless you disable webhook_authentication in Code Manager configuration. | The complete authentication token. |
For example, a GitHub webhook might look like this:

```
https://primary.example.com:8170/code-manager/v1/webhook?type=github&token=$TOKEN
```

A Stash webhook with the optional `prefix` parameter specified might look like:

```
https://primary.example.com/code-manager/v1/webhook?
type=stash&prefix=dev&token=$TOKEN
```

You must attach the complete authentication token to the request. A GitHub request with the entire token attached might look like this:

```
https://primary.example.com:8170/code-manager/v1/webhook?
type=github&token=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkZXBsb3b3IiLCJQIjoiYmxvZ29yeSIyLCAiaWF0IjoxODg0MjY2MjU1LCAidXNlciI6eyJpc3MiOiJkZXBsb3b3IiLCJhbGciOiJJUzI1NiIsImtldCI6IjYiLCJpYXQiOjE2MDM3MjU3MDB9.R-57_l3ziljP2FP-moXQkQK7o7"
```

**Response format**

When you trigger your webhook by pushing changes to your control repository, you can see the response in your Git provider interface. Code Manager does not give a command line response to typical webhook usage.

If you hit the webhook endpoint directly using a curl command with a properly formatted request, valid authentication token, and a valid value for the "type" parameter, Code Manager returns a `{"status":"OK"}` response, whether or not code successfully deployed.

**Error responses**

When you trigger your webhook by pushing changes to your control repository, you can see any errors in your Git provider interface. Code Manager does not give a command line response to typical webhook usage.

If you hit the webhook endpoint directly using a curl command with a request that has an incorrect "type" value or no "type" value, you get an error response such as:

```
{"kind":"puppetlabs.code-manager/unrecognized-webhook-type","msg":"Unrecognized webhook type: 'githubby'","details":"Currently supported valid webhook types include: github gitlab stash tfs-git bitbucket"}
```

**GET /v1/deploys/status**

The Code Manager `/deploys/status` endpoint returns the status of all code deployments that Code Manager is processing for each environment. You can specify an `id` query parameter to return the status of a particular deployment.

**Request format**

The body of the GET `/deploys/status` request must include the authentication token and a URL containing:

- The name of the primary server, such as `primary.example.com`.
- The Code Manager port, such as `8170`.

© 2022 Puppet, Inc., a Perforce company
• The endpoint.

To return the status of all code deployments:

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/status"
curl --insecure --header "$type_header" --header "$auth_header" --header "$type_header" "$uri"
```

To return the status of a specified code deployment:

```bash
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/status?id=1"
curl --insecure --header "$type_header" --header "$auth_header" --header "$type_header" "$uri"
```

See Usage notes for curl examples for information about forming curl commands.

### Response format

The `/deploys/status` endpoint responds with a list of all code deployment requests that Code Manager is processing.

The response contains three sections:

- "deploys-status": Lists the status for each code deployment that Code Manager is processing, including failed deployments. Deployments can be "new", "queued", "deploying", or "failed". Environments that have been successfully deployed to either the code staging or live code directories are displayed in the "file-sync-storage-status" or "file-sync-client-status" sections, respectively.
- "file-sync-storage-status": Lists all environments that Code Manager has successfully deployed to the code staging directory, but not yet synced to the live code directory.
- "file-sync-client-status": Lists status for your primary server and each compiler that Code Manager is deploying environments to, including whether the code in the primary server's staging directory has been synced to its live code directory.

The response can contain the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;queued&quot;</td>
<td>The environment is queued for deployment.</td>
<td>For each deployment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;id&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;queued-at&quot;</td>
</tr>
<tr>
<td>&quot;deploying&quot;</td>
<td>The environment is in the process of being deployed.</td>
<td>For each deployment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;id&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;queued-at&quot;</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
<td>Values</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;new&quot;</td>
<td>Code Manager has accepted a request to deploy the environment, but has not yet queued it.</td>
<td>For each deployment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;id&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;queued-at&quot;</td>
</tr>
<tr>
<td>&quot;failed&quot;</td>
<td>The code deployment for the environment has failed.</td>
<td>For each deployment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;id&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;error&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;details&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;corrected-name&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;kind&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;msg&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;queued-at&quot;</td>
</tr>
<tr>
<td>&quot;deployed&quot;</td>
<td>Lists information for each deployed environment.</td>
<td>For each deployment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;date&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;deploy-signature&quot;</td>
</tr>
<tr>
<td>&quot;all-synced&quot;</td>
<td>Whether all requested code deployments have been synced to the live code directories on their respective servers.</td>
<td>true, false</td>
</tr>
<tr>
<td>&quot;file-sync-clients&quot;</td>
<td>List of all servers that Code Manager is deploying code to.</td>
<td>For each deployment listed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;last_check_in_time&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;synced-with-file-sync-storage&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;deployed&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;environment&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;date&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;deploy-signature&quot;</td>
</tr>
</tbody>
</table>

For example, a complete response without an ID specified might look like this:

```json
{
}
```
"deploys-status":{
  "queued":[
    {
      "environment": "dev",
      "id": 3,
      "queued-at": "2018-05-15T17:42:34.988Z"
    }
  ],
  "deploying":[
    {
      "environment": "test",
      "id": 1,
      "queued-at": "2018-05-15T17:42:34.988Z"
    },
    {
      "environment": "prod",
      "id": 2,
      "queued-at": "2018-05-15T17:42:34.988Z"
    }
  ],
  "new": [
  ],
  "failed": [
  ],
},
"file-sync-storage-status":{
  "deployed": [
    {
      "environment": "prod",
      "date": "2018-05-10T21:44:24.000Z",
      "deploy-signature": "66d620604c9465b464a3dac4884f96c43748b2c5"
    },
    {
      "environment": "test",
      "date": "2018-05-10T21:44:25.000Z",
      "deploy-signature": "24ecc7bac8a4d727d6a3a2350b6fda53812ee86f"
    },
    {
      "environment": "dev",
      "date": "2018-05-10T21:44:21.000Z",
      "deploy-signature": "503a335c99a190501456194d13ff722194e55613"
    }
  ]
},
"file-sync-client-status":{
  "all-synced": false,
  "file-sync-clients":{
    "chihuahua":{
      "last_check_in_time": null,
      "synced-with-file-sync-storage": false,
      "deployed": [
      ]
    },
    "localhost":{
      "synced-with-file-sync-storage": true,
      "deployed": [
        {
          "environment": "prod",
          "date": "2018-05-11T22:40:48.000Z",
        }
      ]
    }
  }
}
The `/deploys/status?id` endpoint responds with details about a specified code deployment.

The response contains three sections:

- "environment" or "project": Type of entity being deployed.
- "status": Status of the deployment, including "queued", "deploying", "new", "failed", "syncing", or "synced".
- "queued-at": Timestamp specifying when the deployment was first queued.

For example, a complete response with an ID specified might look like this:

```
{
  "environment": "production",
  "id": 1,
  "status": "deploying",
  "queued-at": "2018-05-10T21:44:25.000Z"
}
```

### Error responses

If code deployment fails, the "deploys-status" section of the response provides an "error" key for the environment with the failure. The "error" key contains information about the failure.

Deployment failures can remain in the response even after the environment in question is successfully deployed. Old failures are removed from the "deploys-status" response during garbage collection.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;environment&quot;</td>
<td>The name of the environment queued or deployed.</td>
<td>A string specifying the name of an environment.</td>
</tr>
<tr>
<td>&quot;error&quot;</td>
<td>Information about the deployment failure.</td>
<td>Contains keys with error details.</td>
</tr>
<tr>
<td>&quot;corrected-name&quot;</td>
<td>The name of the environment.</td>
<td>A string specifying the name.</td>
</tr>
<tr>
<td>&quot;kind&quot;</td>
<td>The kind of error encountered.</td>
<td>An error type.</td>
</tr>
<tr>
<td>&quot;msg&quot;</td>
<td>The error message.</td>
<td>An error message.</td>
</tr>
<tr>
<td>&quot;queued-at&quot;</td>
<td>The date and time when the environment was queued.</td>
<td>A date and time stamp.</td>
</tr>
</tbody>
</table>
For example, with a failure, the "deploys-status" response section might look something like:

```
[
  {
    "deploys-status": {
      "deploying": [],
      "failed": [
        {
          "environment": "test14",
          "error": {
            "details": {
              "corrected-name": "test14"
            },
            "kind": "puppetlabs.code-manager/deploy-failure",
            "msg": "Errors while deploying environment 'test14' (exit code: 1):
ERROR	 -> Authentication failed for Git remote "https://github.com/puppetlabs/puffppetlabs-apache".
          }
        },
        "new": [],
        "queued": []
      }
    }
  }
]
```

### Managing code with r10k

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to manually deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

Based on the code in your control repository branches, r10k creates environments on your primary server and also installs and updates the modules you want in each environment.

**Tip:** Whenever possible, we recommend Managing code with Code Manager on page 774. Only use r10k if you cannot use Code Manager.
• **Set up r10k** on page 817
  You must set up r10k to use it as your code management tool.

• **Configure r10k** on page 817
  To configure r10k in an existing Puppet Enterprise (PE) installation, set r10k parameters in the PE console. You can also use the console to adjust r10k settings.

• **Customizing r10k configuration** on page 818
  If you need a customized r10k configuration, you can set specific parameters with Hiera.

• **Deploying environments with r10k** on page 825
  Deploy environments on the command line with the `r10k` command.

• **r10k command reference** on page 827
  r10k accepts several command line actions, with options and subcommands.

---

**Set up r10k**

You must set up r10k to use it as your code management tool.

**Tip:** Whenever possible, we recommend Managing code with Code Manager on page 774, because Code Manager does not require manual code deployment. Only use r10k if you cannot use Code Manager.

To set up r10k, you must:

1. Prepare for Managing environments with a control repository on page 764. This involves creating a Git control repository that has a Puppetfile.

   r10k uses the control repo to maintain and deploy your Puppet code and data. You can also create separate deployment environments in your Puppet infrastructure by creating branches in your control repository (such as a development branch for a development environment). r10k tracks your environments and updates them according to the changes you make in your control repo.

   The Puppetfile specifies which modules and data to install in your environment, including what versions to install, and where to download the modules or other content.

2. **Configure r10k** on page 817.

3. Optional: Customize your r10k configuration in Hiera.

4. Use the r10k command line tool to Deploy environments.

**Related information**

Moving from r10k to Code Manager on page 776

Moving from r10k to Code Manager can improve automation of your code management and deployments.

Managing environment content with a Puppetfile on page 768

A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

---

**Configure r10k**

To configure r10k in an existing Puppet Enterprise (PE) installation, set r10k parameters in the PE console. You can also use the console to adjust r10k settings.

**Before you begin**

You need a control repo (with a Puppetfile) and the file path for the SSH private key you created when you set up your control repo. For information and instructions on setting up a control repo, go to Managing environments with a control repository on page 764.

1. In the PE console, go to Node groups > PE Master > Classes, and locate the `puppet_enterprise::profile::master` class.
2. For the \texttt{r10k\_remote} parameter, enter a string that is a valid SSH URL for your Git control repository, such as \texttt{git@<YOUR.GIT.SERVER.COM>:puppet/control.git}.

\textbf{Important:} Some Git providers have additional requirements for enabling SSH access. For example, BitBucket requires \texttt{ssh://} at the beginning of the SSH URL (such as \texttt{ssh://git@<YOUR.GIT.SERVER.COM>:puppet/control.git}). See your provider's documentation for this information.

3. For the \texttt{r10k\_private\_key} parameter, enter a string specifying the path to the SSH private key you created when you set up your control repo, such as \texttt{"/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519"}.

This key permits the \texttt{pe-puppet} user to access your Git control repo. The private key file must be located on the primary server, owned by the \texttt{pe-puppet} user, and in a directory that the \texttt{pe-puppet} user has permission to view. We recommend \texttt{/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519}.

4. Run Puppet on your primary server and compilers.

You can customize your r10k configuration in Hiera, if needed.

To deploy environments with r10k, you must use the command line to manually trigger deployments. PE does not automatically run r10k after you configure it.

\textbf{Related information}

- Configuration parameters and the \texttt{pe.conf} file on page 211

A \texttt{pe.conf} file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default \texttt{pe.conf} file is available in the \texttt{conf.d} directory in the installer tarball.

\textbf{Customizing r10k configuration}

If you need a customized r10k configuration, you can set specific parameters with Hiera.

The customize your configuration:

1. Add parameters to the \texttt{data/common.yaml} file in your control repo. Use the format \texttt{pe\_r10k::<PARAMETER>: <SETTING>}. For example, these parameters specify a Git repo cache directory and the location from which to fetch the source repository:

\begin{verbatim}
pe_r10k::cachedir: /var/cache/r10k
pe_r10k::remote: git://git-server.site/my-org/main-modules
\end{verbatim}

Refer to the other sections on this page for information about other settings you might want to configure.

2. Run Puppet on the primary server.

3. Deploy environments with r10k. PE does not automatically run r10k after you configure it.

\textbf{Related information}

- Configure settings with Hiera on page 287

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

\textbf{Configuring Forge settings}

To configure how r10k downloads modules from the Forge, specify \texttt{forge\_settings} in Hiera.

This parameter configures where Forge modules are installed from, and sets a proxy for all Forge interactions. The \texttt{forge\_settings} parameter accepts a hash with the following values:

- \texttt{baseurl}: Indicates where Forge modules are installed from. Defaults to \texttt{https://forgeapi.puppetlabs.com}.
- \texttt{authorization\_token}: Specifies the token for authenticating to a custom Forge server.
• **proxy**: Sets the proxy for all Forge interactions.

**baseurl**

Indicates where Forge modules are installed from. Defaults to [https://forgeapi.puppetlabs.com](https://forgeapi.puppetlabs.com).

```yaml
pe_r10k::forge_settings:
    baseurl: 'https://private-forge.mysite'
```

**proxy**

Sets the proxy for all Forge interactions. This setting overrides the global proxy setting on Forge operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. See the global proxy setting for more information and examples.

```yaml
puppet_enterprise::master::code_manager::forge_settings:
    proxy: 'http://proxy.example.com:3128'
```

**Configuring purge levels**

The `purge_levels` setting controls which unmanaged content r10k purges during a deployment.

This setting accepts an array of strings specifying what content r10k purges during code deployments. Available values are:

- deployment
- environment
- puppetfile
- purge_allowlist

The default value for this setting is `['deployment', 'puppetfile']`. With these values, r10k purges:

- Any content that is not managed by the sources declared in the `remote` or `sources` settings.
- Any content that is not declared in the Puppetfile, but is found in a directory managed by the Puppetfile.

```yaml
deploy:
    purge_levels: ['deployment', 'environment', 'puppetfile']
```

**deployment**

After each deployment, in the configured basedir, r10k recursively removes content that is not managed by any of the sources declared in the `remote` or `sources` parameters.

Disabling this level of purging could cause the number of deployed environments to grow without bound, because deleting branches from a control repo would no longer cause the matching environment to be purged.

**environment**

After a given environment is deployed, r10k recursively removes content that is neither committed to the control repo branch that maps to that environment, nor declared in a Puppetfile committed to that branch.

With this purge level, r10k loads and parses the Puppetfile for the environment even if the `--modules` flag is not set. This allows r10k to check whether or not content is declared in the Puppetfile. However, Puppetfile content is deployed only if the environment is new or the `--modules` flag is set.

If r10k encounters an error while evaluating the Puppetfile or deploying its contents, no environment-level content is purged.
puppetfile

After Puppetfile content for a given environment is deployed, r10k recursively removes content in any directory managed by the Puppetfile, if that content is not also declared in that Puppetfile.

Directories considered to be managed by a Puppetfile include the configured moduledir (which defaults to "modules"), as well as any alternate directories specified as an install_path option to any Puppetfile content declarations.

purge_allowlist

The purge_allowlist setting exempts the specified filename patterns from being purged.

This setting affects environment level purging only. Any given value must be a list of shell style filename patterns in string format.

See the Ruby documentation for the fnmatch method for more details on valid patterns. Both the FNM_PATHNAME and FNM_DOTMATCH flags are in effect when r10k considers the whitelist.

Patterns are relative to the root of the environment being purged and, by default, **do not match recursively**. For example, a whitelist value of *myfile* would preserve only a matching file at the root of the environment. To preserve the file throughout the deployed environment, you would need to use a recursive pattern such as **/*myfile*.

Files matching a whitelist pattern might still be removed if they exist in a folder that is otherwise subject to purging. In this case, use an additional whitelist rule to preserve the containing folder.

```bash
deploy:
  purge_allowlist: [ 'custom.json', '**/*.xpp' ]
```

Locking r10k deployments

The deploy: write_lock setting allows you to temporarily disallow r10k code deploys without completely removing the r10k configuration.

This deploy subsetting is useful to prevent r10k deployments at certain times or to prevent deployments from interfering with a common set of code that might be touched by multiple r10k configurations.

```bash
deploy:
  write_lock: "Deploying code is disallowed until the next maintenance window."
```

Configuring sources

If you are managing more than one repository with r10k, specify a map of your source repositories.

Use the source parameter to specify a map of sources. Configure this setting if you are managing more than just Puppet environments, such as when you are also managing Hiera data in its own control repository.

To use multiple control repos, the sources setting and the repositories setting must match.

If sources is set, you cannot use the global remote and r10k_basedir settings. Note that r10k raises an error if different sources collide in a single base directory. If you are using multiple sources, use the prefix option to prevent collisions.

This setting accepts a hash with the following subsettings:

- remote
- basedir
- prefix
- invalid_branches

```bash
myorg:
  remote: "git://git-server.site/myorg/main-modules"
```
remote

Specifies where the source repository should be fetched from. The remote must be able to be fetched without any interactive input. That is, you cannot be prompted for usernames or passwords in order to fetch the remote.

Accepts a string that is any valid URL that r10k can clone, such as:

```
git://git-server.site/my-org/main-modules
```

basedir

Specifies the path where environments are created for this source. This directory is entirely managed by r10k, and any contents that r10k did not put there are removed.

Note that the `basedir` setting must match the `environmentpath` in your puppet.conf file, or Puppet won't be able to access your new directory environments.

prefix

Allows environment names to be prefixed with this string. Alternatively, if `prefix` is set to true, the source's name is used. This prevents collisions when multiple sources are deployed into the same directory.

In the `sources` example above, two "main-modules" environments are set up in the same base directory. The prefix setting is used to differentiate the environments: the first is named "myorg-main-modules", and the second is "testing-main-modules".

ignore_branch_prefixes

Causes branches from a source which match any of the prefixes listed in the setting to be ignored. The match can be full or partial. On deploy, each branch in the repo has its name tested against all prefixes listed in `ignore_branch_prefixes` and, if the prefix is found, then an environment is not deployed for this branch.

The setting is a list of strings. In the following example, branches in "mysource" with the prefixes "test" and "dev" are not deployed.

```
---
sources:
  mysource:
    basedir: '/etc/puppet/environments'
  ignore_branch_prefixes:
    - 'test'
    - 'dev'
```

If `ignore_branch_prefixes` is not specified, then all branches are deployed.

invalid_branches

Specifies how branch names that cannot be cleanly mapped to Puppet environments are handled. This option accepts three values:

- `'correct_and_warn'` is the default value and replaces non-word characters with underscores and issues a warning.
• 'correct' replaces non-word characters with underscores without warning.
• 'error' ignores branches with non-word characters and issues an error.

**Configuring the r10k base directory**
The r10k base directory specifies the path where environments are created for this source.

This directory is entirely managed by r10k, and any contents that r10k did not put there are removed. If `r10k_basedir` is not set, it uses the default `environmentpath` in your `puppet.conf` file. If `r10k_basedir` is set, you cannot set the `sources` parameter.

Note that the `r10k_basedir` setting **must match** the `environmentpath` in your `puppet.conf` file, or Puppet can't access your new directory environments.

Accepts a string, such as: `/etc/puppetlabs/code/environments`.

**Configuring r10k Git settings**
To configure r10k to use a specific Git provider, a private key, a proxy, or multiple repositories with Git, specify the `git_settings` parameter.

The r10k `git_settings` parameter accepts a hash of the following settings: Contains settings specific to Git.

- `private_key`
- `proxy`
- `repositories`
- `provider`

Accepts a hash of values, such as:

```yaml
pe_r10k::git_settings:
  provider: "rugged"
  private_key: "/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519"
  username: "git"
```

**private_key**
Specifies the file containing the default private key used to access control repositories, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`. This file must have read permissions for the `pe-puppet` user. The SSH key cannot require a password. This setting is required, but by default, the value is supplied in the `puppet-enterprise::profile::master` class.

**proxy**
Sets a proxy specifically for Git operations that use an HTTP(S) transport.

This setting overrides the global `proxy` setting on Git operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. To set a proxy for a specific Git repository only, set `proxy` in the `repositories` subsetting of `git_settings`. See the global `proxy` setting for more information and examples.

```yaml
proxy: 'http://proxy.example.com:3128'
```

**repositories**
Specifies a list of repositories and their respective private keys. Use this setting if you want to use multiple control repos.

To use multiple control repos, the `sources` setting and the `repositories` setting must match. Accepts an array of hashes with the following keys:

- `remote`: The repository these settings should apply to.
• **private_key**: The file containing the private key to use for this repository. This file must have read permissions for the pe-puppet user. This setting overrides the global `private_key` setting.

• **proxy**: The proxy setting allows you to set or override the global proxy setting for a single, specific repository. See the global `proxy` setting for more information and examples.

```yaml
pe_r10k::git_settings:
  repositories:
    - remote: "ssh://tessier-ashpool.freeside/protected-repo.git"
      private_key: "/etc/puppetlabs/r10k/ssh/id_rsa-protected-repo-deploy-key"
    - remote: "https://git.example.com/my-repo.git"
      proxy: "https://proxy.example.com:3128"
```

**username**

Optionally, specify a username when the Git remote URL does not provide a username.

**Configuring proxies**

To configure proxy servers, use the `proxy` setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

**proxy**

To set a proxy for all operations occurring over an HTTP(S) transport, set the global `proxy` setting. You can also set an authenticated proxy with either Basic or Digest authentication.

To override this setting for Git or Forge operations only, set the `proxy` subsetting under the `git_settings` or `forge_settings` parameters. To override for a specific Git repository, set a proxy in the `repositories` list of `git_settings`. To override this setting with no proxy for Git, Forge, or a particular repository, set that specific `proxy` setting to an empty string.

In this example, the first proxy is set up without authentication, and the second proxy uses authentication:

```yaml
proxy: 'http://proxy.example.com:3128'
proxy: 'http://user:password@proxy.example.com:3128'
```

**r10k proxy defaults and logging**

By default, r10k looks for and uses the first environment variable it finds in this list:

- `HTTPS_PROXY`
- `https_proxy`
- `HTTP_PROXY`
- `http_proxy`

If no proxy setting is found in the environment, the global `proxy` setting defaults to no proxy.

The proxy server used is logged at the "debug" level when r10k runs.

**Configuring postrun commands**

To configure a command to be run after all deployment is complete, add the `postrun` parameter.

The `postrun` optional command to be run after r10k finishes deploying environments. The command must be an array of strings to be used as an argument vector. You can set this parameter only one time.

```bash
postrun: ['/usr/bin/curl', '-F', 'deploy=done', 'http://my-app.site/endpoint']
```
**r10k parameters**
The following parameters are available for r10k. Parameters are optional unless otherwise stated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cachedir</td>
<td>Specifies the path to the directory where you want the cache of your Git repos to be stored.</td>
<td>A valid directory path</td>
<td>/var/cache/r10k</td>
</tr>
<tr>
<td>deploy</td>
<td>Controls how r10k code deployments behave.</td>
<td>Accepts settings for:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Configuring purge levels and Locking deployments for usage details.</td>
<td>• purge_level</td>
<td>• purge_level:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• write_lock</td>
<td>• 'deployment',</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 'puppetfile'</td>
</tr>
<tr>
<td>forge_settings</td>
<td>Contains settings for downloading modules from the Puppet Forge. See Configuring Forge settings for usage details.</td>
<td>Accepts a hash of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• baseurl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• proxy</td>
<td></td>
</tr>
<tr>
<td>git_settings</td>
<td>Configures Git settings: Git provider, private key, proxies, and repositories. See Configuring Git settings for usage details.</td>
<td>Accepts a hash of:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• provider</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• private_key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• proxy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• repositories</td>
<td></td>
</tr>
<tr>
<td>proxy</td>
<td>Configures a proxy server to use for all operations that occur over an HTTP(S) transport. See Configuring proxies for usage details.</td>
<td>Accepts a string specifying a URL to proxy server, without authentication, or with Basic or Digest authentication.</td>
<td></td>
</tr>
<tr>
<td>postrun</td>
<td>An optional command to be run after r10k finishes deploying environments.</td>
<td>An array of strings to use as an argument vector.</td>
<td></td>
</tr>
<tr>
<td>remote</td>
<td>Specifies where the source repository should be fetched from. The remote cannot require any prompts, such as for usernames or passwords. If remote is set, you cannot use sources.</td>
<td>Accepts a string that is any valid SSH URL that r10k can clone.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, uses the r10k_remote value set in the console.</td>
<td></td>
</tr>
</tbody>
</table>
### Deploying environments with r10k

Deploy environments on the command line with the `r10k` command.

The first time you run `r10k`, deploy all environments and modules by running:

```
r10k deploy environment
```

This command:

1. Checks your control repository to see which branches are present.
2. Maps those branches to the Puppet directory environments.
3. Clones your Git repo, and either creates (if this is your first run) or updates (if it is a subsequent run) your directory environments with the contents of your repo branches.

**Restriction:** When running commands to deploy code on your primary server, `r10k` needs write access to your environment path. Run `r10k` as the `pe-puppet` user, as root, or use `sudo`. Running as root requires access control to the root user.

**Related information**

- `r10k command reference` on page 827
- `r10k` accepts several command line actions, with options and subcommands.

### Updating environments

To update environments with `r10k`, use the `deploy environment` command.

**Updating all environments**

The `deploy environment` command updates all existing environments and recursively creates new environments.

This command updates modules only on the first deployment of a given environment. On subsequent updates, it only updates the environment.

From the command line, run `r10k deploy environment`

**Updating all environments and modules**

With the `--modules` flag, the `deploy environment` command updates all environments and modules.

This command:

- Updates all sources.
- Creates new environments.
- Deletes old environments.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r10k_basedir</code></td>
<td>Specifies the path where environments are created for this source. See Configuring the base directory for usage details.</td>
<td>A valid file path, which must match the value of <code>environmentpath</code> in your <code>puppet.conf</code> file.</td>
<td>Uses the value of the <code>environmentpath</code> in your <code>puppet.conf</code> file.</td>
</tr>
<tr>
<td><code>sources</code></td>
<td>Specifies a map of sources to be passed to r10k. Use if you are managing more than just Puppet environments. See Configuring sources for usage details.</td>
<td>A hash of: • <code>basedir</code> • <code>remote</code> • <code>prefix</code> • <code>ignore_branch_prefixes</code></td>
<td>No default.</td>
</tr>
</tbody>
</table>
• Recursively updates all environment modules specified in each environment's Puppetfile.

This command does the maximum possible work, and is therefore the slowest method for r10k deployments. Usually, you must use the less resource-intensive commands for updating environments and modules.

From the command line, run `r10k deploy environment --modules`

**Updating a single environment**
To update a single environment, specify the environment name with the `deploy environment` command.

This command updates modules only on the first deployment of the specified environment. On subsequent updates, it only updates the environment.

If you are actively developing on a given environment, running the following command is the quickest way to deploy your changes:

`r10k deploy environment <MY_WORKING_ENVIRONMENT>`

**Updating a single environment and its modules**
To update both a single given environment and all of the modules contained in that environment's Puppetfile, add the `--modules` flag to your command. This is useful if you want to make sure that a given environment is fully up to date.

On the command line, run `r10k deploy environment <MY_WORKING_ENVIRONMENT> --modules`

**Installing and updating modules**
To update modules, use the `r10k deploy module` subcommand. This command installs or updates the modules specified in each environment's Puppetfile.

If the specified module is not described in a given environment's Puppetfile, that environment is skipped.

**Updating specific modules across all environments**
To update specific modules across all environments, specify the modules with the `deploy module` command.

**Important:** You must run the `r10k deploy environment` command without the `--modules` flag before you update modules across all environments. This command fetches any updates to the applicable Puppetfile.

You can specify a single module or multiple modules. For example, to update the apache, jenkins, and java modules, run `r10k deploy module apache jenkins java`

**Updating one or more modules in a single environment**
To update specific modules in a single environment, specify both the environment and the modules.

**Important:** You must run the `r10k deploy <environment>` command without the `--modules` flag before you update modules in a single environment. This command fetches any updates to the applicable Puppetfile.

On the command line, run `r10k deploy module` with:

• The environment option `-e`.
• The name of the environment.
• The names of the modules you want to update in that environment.

For example, to install the apache, jenkins, and java modules in the production environment, run `r10k deploy module -e production apache jenkins java`

**Viewing environments with r10k**
Display information about your environments and modules with the `r10k deploy display` subcommand. This subcommand does not deploy environments, but only displays information about the environments and modules r10k is managing.

This command can return varying levels of detail about the environments.

• To display all environments being managed by r10k, use the `deploy display` command. From the command line, run `r10k deploy display`
• To display all managed environments and Puppetfile modules, use the Puppetfile flag, -m. From the command line, run `r10k deploy display -m`.

• To get detailed information about module versions, use the -m (Puppetfile) and --detail flags. This provides both the expected and actual versions of the modules listed in each environment's Puppetfile. From the command line, run `r10k deploy display -m --detail`.

• To get detailed information about the modules only in specific environments, limit your query with the environment names. This provides both the expected and actual versions of the modules listed in the Puppetfile in each specified environment. For example, to see details on the modules for environments called production, vmwr, and webrefactor, run:

```
r10k deploy display -m --detail production vmwr webrefactor
```

### r10k command reference

r10k accepts several command line actions, with options and subcommands.

#### r10k command actions

r10k includes several command line actions.

<table>
<thead>
<tr>
<th>r10k command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>deploy</td>
<td>Performs operations on environments. Accepts deploy subcommands listed below.</td>
</tr>
<tr>
<td>help</td>
<td>Displays the r10k help page in the terminal.</td>
</tr>
<tr>
<td>puppetfile</td>
<td>Performs operations on a Puppetfile. Accepts puppetfile subcommands.</td>
</tr>
<tr>
<td>version</td>
<td>Displays your r10k version in the terminal.</td>
</tr>
</tbody>
</table>

#### r10k deploy subcommands

You can use the following subcommands with the r10k deploy command.

```
r10k deploy --display
```

<table>
<thead>
<tr>
<th>deploy subcommand</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Displays a list of environments in your deployment.</td>
</tr>
<tr>
<td>display -m</td>
<td>Displays a list of modules in the Puppetfile.</td>
</tr>
<tr>
<td>display --detail</td>
<td>Displays detailed information.</td>
</tr>
<tr>
<td><strong>deploy subcommand</strong></td>
<td><strong>Action</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>display --fetch</td>
<td>Update the environment sources. Allows you to check for missing environments.</td>
</tr>
<tr>
<td>environment</td>
<td>Deploys environments and their specified modules.</td>
</tr>
<tr>
<td>environment -m</td>
<td>Updates modules specified in the environment's Puppetfile.</td>
</tr>
<tr>
<td>environment -m --incremental</td>
<td>Install or update only modules set to a &quot;floating&quot; version and any modules whose definitions are new or changed since the last deployment.</td>
</tr>
<tr>
<td>module</td>
<td>Deploys a module in all environments.</td>
</tr>
<tr>
<td>module -e</td>
<td>Updates all modules in a particular environment.</td>
</tr>
<tr>
<td>no-force</td>
<td>Prevents the overwriting of local changes to Git-based modules.</td>
</tr>
</tbody>
</table>

**Note:** The --p subcommand option is deprecated. Use --m instead.

See the related topic about deploying environments with r10k for examples of r10k deploy command use.

**r10k puppetfile subcommands**

The r10k puppetfile command accepts several subcommands for managing modules.

r10k puppetfile install

These subcommands must be run as the user with write access to that environment’s modules directory. These commands interact with the Puppetfile in the current working directory, so before running the subcommand, make sure you are in the directory of the Puppetfile you want to use.

<table>
<thead>
<tr>
<th><strong>puppetfile subcommand</strong></th>
<th><strong>Action</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>check</td>
<td>Verifies the Puppetfile syntax is correct.</td>
</tr>
<tr>
<td>install</td>
<td>Installs all modules from a Puppetfile.</td>
</tr>
<tr>
<td>install --modules</td>
<td>Installs modules from a custom Puppetfile path.</td>
</tr>
<tr>
<td>install --moduledir</td>
<td>Installs modules from a Puppetfile to a custom module directory location.</td>
</tr>
<tr>
<td>install --update_force</td>
<td>Installs modules from a Puppetfile with a forced overwrite of local changes to Git-based modules.</td>
</tr>
<tr>
<td>purge</td>
<td>Purges unmanaged modules from a Puppetfile managed directory.</td>
</tr>
</tbody>
</table>

**Note:** The --p puppetfile subcommand option is deprecated. Use --modules instead.

---

**About file sync**

File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

When triggered by a web endpoint, file sync takes changes from your working directory on your primary server and deploys the code to a live code directory. File sync then automatically deploys that code to all compilers.
In addition, file sync ensures that your Puppet code is deployed only when it is ready. These deployments ensure that your agents’ code do not change during a run. File sync also triggers an environment cache flush when the deployment has finished, to ensure that new agent runs happen against the newly deployed Puppet code.

**Important:**

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync. Directly accessing the file sync API is unsupported and not recommended.

For information about how code deployments are triggered, refer to How Code Manager works on page 774.

**Tip:** If you want code deployments to run uninterrupted while Orchestrator plans are running, you can enable Running plans alongside code deployments on page 669.

---

**File sync terms**

There are a few terms that are helpful when you are working with file sync or tools that use file sync.

**Live code directory**

This is the directory that all of the Puppet environments, manifests, and modules are stored in. This directory is used by Puppet Server for catalog compilation. It corresponds to the Puppet Server `master-code-dir` setting and the Puppet `$codedir` setting. The default value is `/etc/puppetlabs/code`. In file sync configuration, you might see this referred to simply as live-dir. This directory exists on your primary server and compilers.

**Staging code directory**

This is the directory where you stage your code changes before rolling them out to the live code dir. You can move files into this directory in your usual way. Then, when you trigger a file sync deployment, file sync moves the changes to the live code dir on your primary server and compilers. This directory exists only on the primary server; compilers do not need a staging directory. The default value is `/etc/puppetlabs/code-staging`.

---

**How file sync works**

File sync helps distribute your code to your primary server, compilers, and agents.

By default, file sync is disabled and the staging directory is not created on the primary server. If you're upgrading from 2015.2 or earlier, file sync is disabled after the upgrade. You must enable file sync, and then run Puppet on your primary server and compilers. This creates the staging directory on the primary server, which you can then populate with your Puppet code. File sync can then commit your code; that is, it can prepare the code for synchronization to the live code directory, and then your compilers. Normally, Code Manager triggers this commit automatically, but, only if absolutely necessary, you can trigger a commit by hitting the file sync endpoint.

**Important:**

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync. File sync endpoint documentation is for informational purposes only. Directly accessing the file sync API is unsupported and not recommended.

For example:

```bash
type_header='Content-Type: application/json'
cert='$(puppet config print hostcert)'
cacert='$(puppet config print localcacert)'
key='$(puppet config print hostprivkey)'
uri='https://$(puppet config print server):8140/file-sync/v1/commit'
data='{"commit-all": true}''

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key" --request POST "$uri" --data "$data"
```

© 2022 Puppet, Inc., a Perforce company
See Usage notes for curl examples for information about forming curl commands.

The above command is run from the primary server and contains the following variables:

- **fqdn**: Fully qualified domain name of the primary server.
- **cert**: The ssl cert for the primary server.
- **cacert**: The Puppet CA's certificate.
- **key**: The private key for the primary server.

This command commits all of the changes in the staging directory. After the commit, when any compilers check the file sync service for changes, they receive the new code and deploy it into their own code directories, where it is available for agents checking in to those compilers. (By default, compilers check file sync every 5 seconds.) Commits can be restricted to a specific environment and can include details such as a message, and information about the commit author.

**Enabling or disabling file sync**

File sync is normally enabled or disabled automatically along with Code Manager.

File sync's behavior is linked to that of Code Manager. Because Code Manager is disabled by default, file sync is also disabled. To enable file sync, enable Code Manager. You can enable and configure Code Manager either during or after PE installation.

The `file_sync_enabled` parameter in the `puppet_enterprise::profile::master` class in the console defaults to `automatic`, which means that file sync is enabled and disabled automatically with Code Manager. If you set this parameter to `true`, it forces file sync to be enabled even if Code Manager is disabled. The `file_sync_enabled` parameter doesn't appear in the class definitions --- you must add the parameter to the class in order to set it.

**Resetting file sync**

If file sync has entered a failure state, consumed all available disk space, or a repository has become irreparably corrupted, reset the service.

Resetting deletes the commit history for all repositories managed by file sync, which frees up disk space and returns the service to a "fresh install" state while preserving any code in the staging directory.

1. On the primary server, perform the appropriate action:
   - If you use file sync with Code Manager, ensure that any code ready for deployment is present in the staging directory, and that the code most recently deployed is present in your control repository so that it can be re-synced.
   - If you use file sync with r10k, perform an r10k deploy and ensure that the code most recently deployed is present in your control repository so that it can be re-synced.
   - If you use file sync alone, ensure that any code ready for deployment is present in `/etc/puppetlabs/code-staging`.
2. Shut down the `pe-puppetserver` service: `puppet resource service pe-puppetserver ensure=stopped`.
3. Delete the data directory located at `/opt/puppetlabs/server/data/puppetserver/filesync/storage`.
4. Restart the `pe-puppetserver` service: `puppet resource service pe-puppetserver ensure=running`.
5. Perform the appropriate action:
   - If you use file sync with Code Manager, use Code Manager to deploy all environments.
   - If you use file sync alone or with r10k, perform a commit.

File sync is now reset. The service creates fresh repositories on each client and the storage server for the code it manages.
Checking your deployments

You can manually look up information about file sync's deployments by using curl commands that hit the status endpoint.

Important:

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync.

File sync endpoint documentation is for informational purposes only. Directly accessing the file sync API is unsupported and not recommended.

To look up deployment information, run the following curl command:

```
curl --insecure "https://$(puppet config print server):8140/status/v1/services/file-sync-storage-service?level=debug"
```

The curl command returns a list of:

- All of the clients that file sync is aware of.
- When those clients last checked in.
- Which commit they have deployed.

Note: This command returns output in JSON, so you can pipe it to `python -m json.tool` for better readability. See Using example commands on page 28 for information about forming curl commands.

You can check if a specific commit was deployed by viewing the `latest_commit`. This commit is listed as the `latest_commit`.

Important: The SHA reported as `latest_commit` is internal to file sync and not from the user's control repository.

Cautions

There are a few things to be aware of with file sync.

File sync API

Because file sync is part of Code Manager, Code Manager handles communication with the file sync API.

Information about the file sync API in this documentation is for informational purposes only.

Important: Do not directly access the file sync API.

Where to edit code

With Code Manager, you only modify code in your control repo. Changes made in invalid locations are overwritten by the next deployment. For more information, refer to Understanding file sync and the staging directory on page 776.

While extremely uncommon, if you're using file sync without Code Manager, only modify your Puppet code in the staging directory.

The `enable-forceful-sync` parameter is set to `true` by default in PE. When `false`, file sync no longer overwrites changes in the live code directory. Instead, it logs errors to the Puppet Server log (at `/var/log/puppetlabs/puppetserver/puppetserver.log`).

If you need to set this parameter to `false`, you must add it with Hiera:

```
puppet_enterprise::master::file_sync::file_sync_enable_forceful_sync: false
```
The puppet module command and file sync

The puppet module command doesn't work with file sync. You'll need to specify modules in your environment's Puppetfiles and use Code Manager to handle module code deployments. For information and instructions, refer to:

- Managing modules with a Puppetfile on page 769
- Managing code with Code Manager on page 774

Permissions

File sync runs as the pe-puppet user. To sync files, file sync must have permission to read the staging directory and to write to all files and directories in the live code directories. You can run the following command to make sure the pe-puppet user owns the required directories:

```
chown -R pe-puppet /etc/puppetlabs/code /etc/puppetlabs/code-staging
```

Environment isolation metadata

File sync generates .pp metadata files in your staging code directory and live code directories. These files provide environment isolation for your resource types, which ensures that each environment uses the correct version of the resource type.

⚠️ CAUTION: Do not delete or modify these files. Do not use expressions from these files in regular manifests.

For more details about these files and how they isolate resource types in multiple environments, refer to Environment isolation in the Puppet documentation. For information about when these files are generated, refer to Environment isolation metadata and Code Manager on page 776.

SSL and certificates

Network communications and security in Puppet Enterprise are based on HTTPS, which secures traffic using X.509 certificates. PE includes its own CA tools, which you can use to regenerate certs as needed.

- Regenerate the console certificate on page 833
  The console certificate expires every 824 days. Regenerate the console certificate when it's nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.
- Regenerate infrastructure certificates on page 833
  Regenerating certificates and security credentials—both private and public keys—created by the built-in PE certificate authority can help ensure the security of your installation in certain cases.
- Use an independent intermediate certificate authority on page 836
  The built-in Puppet certificate authority automatically generates a root and intermediate certificate, but if you need additional intermediate certificates or prefer to use a public authority CA, you can set up an independent intermediate certificate authority. You must complete this configuration during installation.
- Use a custom SSL certificate for the console on page 838
  The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.
• **Generate a custom Diffie-Hellman parameter file** on page 840
  The "Logjam Attack" (CVE-2015-4000) exposed several weaknesses in the Diffie-Hellman (DH) key exchange. To help mitigate the "Logjam Attack," PE ships with a pre-generated 2048 bit Diffie-Hellman param file. In the case that you don't want to use the default DH param file, you can generate your own.

• **Enable TLSv1** on page 840
  TLSv1 and TLSv1.1 are disabled by default in PE.

---

**Regenerate the console certificate**

The console certificate expires every 824 days. Regenerate the console certificate when it's nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.

After you delete the existing console certificate and its artifacts, a new certificate is generated automatically on the next Puppet run.

1. On your primary server, run:

   ```bash
   rm /etc/puppetlabs/puppet/ssl/ca/signed/console-cert.pem /
   etc/puppetlabs/puppet/ssl/certs/console-cert.pem /
   etc/puppetlabs/puppet/ssl/private_keys/
   console-cert.pem /etc/puppetlabs/puppet/ssl/public_keys/console-cert.pem
   ```

2. Run Puppet, `puppet agent -t` or wait for the next Puppet run.

---

**Regenerate infrastructure certificates**

Regenerating certificates and security credentials—both private and public keys—created by the built-in PE certificate authority can help ensure the security of your installation in certain cases.

The process for regenerating certificates varies depending on your goal.

<table>
<thead>
<tr>
<th>If your goal is to...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upgrade to the intermediate certificate architecture introduced in Puppet 6.0.</td>
<td>Complete these tasks in order:</td>
</tr>
<tr>
<td>Fix a compromised or damaged certificate authority.</td>
<td>1. Delete and recreate the certificate authority on page 834</td>
</tr>
<tr>
<td></td>
<td>2. Regenerate compiler certificates on page 834, if applicable</td>
</tr>
<tr>
<td></td>
<td>3. Regenerate agent certificates on page 834</td>
</tr>
<tr>
<td></td>
<td>4. Regenerate replica certificates on page 835</td>
</tr>
<tr>
<td>Fix a compromised compiler certificate or troubleshoot SSL errors on compilers.</td>
<td>Regenerate compiler certificates on page 834</td>
</tr>
<tr>
<td>Fix a compromised agent certificate or troubleshoot SSL errors on agent nodes.</td>
<td>Regenerate agent certificates on page 834</td>
</tr>
<tr>
<td>Specify a new DNS alt name or other trusted data.</td>
<td>Regenerate primary server certificates on page 835</td>
</tr>
</tbody>
</table>

**Note:** To support recovery, backups of your certificates are saved and the location of the backup directory is output to the console. If the command fails after deleting the certificates, you can restore your certificates with the contents of this backup directory.
Delete and recreate the certificate authority

Recreate the certificate authority only if you're upgrading to the new certificate architecture introduced in Puppet 6.0, or if your certificate authority was compromised or damaged beyond repair.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see Bolt OpenSSH configuration options.

**CAUTION:** Replacing your certificate authority invalidates all existing certificates in your environment. Complete this task only if and when you're prepared to regenerate certificates for both your infrastructure nodes (including external PE-PostgreSQL in extra-large installations) and your entire agent fleet.

On your primary server, log in as root, and run:

```bash
puppet infrastructure run rebuild_certificate_authority
```

The SSL and cert directories on your CA server are backed up with "_bak" appended to the end, CA files are removed and certificates are rebuilt, and a Puppet run completes.

Regenerate compiler certificates

Regenerate compiler certificates to fix a compromised certificate or troubleshoot SSL errors on compilers, or if you recreated your certificate authority.

On your primary server, log in as root and run the following command. Specify any additional parameters required for your environment and use case.

```bash
puppet infrastructure run regenerate_compiler_certificate
target=<COMPILER_HOSTNAME>
```

- **If you use DNS alternative names**, specify `dns_alt_names` as a comma-separated list of names to add to agent certificates.

  **Important:** To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- **If you recreated your certificate authority**, or are otherwise unable to connect to the compiler with the orchestrator, specify `--use-ssh` any any additional parameters needed to connect over SSH.

  The compiler's SSL directory is backed up to `/etc/puppetlabs/puppet/ssl_bak`, its certificate is regenerated and signed, a Puppet run completes, and the compiler resumes its role in your deployment.

Regenerate agent certificates

Regenerate *nix or Windows agent certificates to fix a compromised certificate or troubleshoot SSL errors on agents, or if you recreated your certificate authority.

On your primary server, log in as root and run the following command. Specify any additional parameters required for your environment and use case.

```bash
puppet infrastructure run regenerate_agent_certificate
agent=<AGENT_HOSTNAME_1>,<AGENT_HOSTNAME_2>
```
• If you use DNS alternative names, specify dns_alt_names as a comma-separated list of names to add to agent certificates.

**Important:** To ensure naming consistency, if your puppet.conf file includes a dns_alt_names entry, you must include the dns_alt_names parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

• If you recreated your certificate authority, specify clean_crl=true and --use-ssh, as well as any additional parameters needed to connect over SSH.

Agent SSL directories are backed up to /etc/puppetlabs/puppet/ssl_bak (*nix) or C:/ProgramData/PuppetLabs/puppet/etc/ssl_bak (Windows), their certificates are regenerated and signed, a Puppet run completes, and the agents resume their role in your deployment.

**Regenerate replica certificates**

Regenerate replica certificates for your disaster recovery installation to specify a new DNS alt name or other trusted data, or if you recreated your certificate authority.

On your primary server, log in as root and run the following command. Specify any additional parameters required for your environment and use case.

```bash
puppet infrastructure run regenerate_replica_certificate
target=<REPLICA_HOSTNAME>
```

• If you use DNS alternative names, specify dns_alt_names as a comma-separated list of names to add to agent certificates.

**Important:** To ensure naming consistency, if your puppet.conf file includes a dns_alt_names entry, you must include the dns_alt_names parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

• If you recreated your certificate authority, or are otherwise unable to connect to the replica with the orchestrator, specify --use-ssh and any additional parameters needed to connect over SSH.

The replica's SSL directory is backed up to /etc/puppetlabs/puppet/ssl_bak, its certificate is regenerated and signed, a Puppet run completes, and the replica resumes its role in your deployment.

**Regenerate primary server certificates**

Regenerate primary server certificates to specify a new DNS alt name or other trusted data. This process regenerates the certificates for all primary infrastructure nodes, including standalone PE-PostgreSQL nodes.

**Before you begin**

The puppet infrastructure run command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in .ssh/config on your primary server. For more information, see Bolt OpenSSH configuration options.

To view all available parameters, use the --help flag. The logs for all puppet infrastructure run Bolt plans are located at /var/log/puppetlabs/installer/bolt_info.log.

On your primary server, log in as root and run:

```bash
puppet infrastructure run regenerate_master_certificate
```

You can specify these optional parameters:
• Use `dns_alt_names` to provide a comma-separated list of alternate DNS names to be added to the certificates generated for your primary server.

**Important:** To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your primary server certificate.

• Use `--tmpdir` to specify a path to a directory to use for uploading and executing temporary files.
• Use `--force` to force certificate regeneration in situations where your infrastructure is unhealthy due to a damaged certificate.

---

## Use an independent intermediate certificate authority

The built-in Puppet certificate authority automatically generates a root and intermediate certificate, but if you need additional intermediate certificates or prefer to use a public authority CA, you can set up an independent intermediate certificate authority. You must complete this configuration during installation.

---

**CAUTION:** This method requires more manual maintenance than the default certificate authority setup. With an external chain of trust, you must monitor for and promptly update expired CRLs, because an expired CRL anywhere in the chain causes certificate validation failures. To manage an external CRL chain:

- Take note of the `Next Update` dates of the CRLs for your entire chain of trust.
- Submit updated CRLs to Puppet Server using the `certificate_revocation_list` endpoint.
- Configure agents to download CRL updates by setting `crl_refresh_interval` in the `puppet_enterprise::profile::agent` class.
1. Collect the PEM-encoded certificates and CRLs for your organization's chain of trust, including the root certificate, any intermediate certificates, and the signing certificate. (The signing certificate might be the root or intermediate certificate.)

2. Create a private RSA key, with no passphrase, for the Puppet CA.

3. Create a PEM-encoded Puppet CA certificate.
   a) Create a CSR for the Puppet CA.
   b) Generate the Puppet CA certificate by signing the CSR using your external CA.

   Ensure the CA constraint is set to true and the keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the bit string subjectPublicKeyfield. See RFC 5280 section 4.2.1.2 for details.

4. Concatenate all of the certificates into a PEM-encoded certificate bundle, starting with the Puppet CA cert and ending with your root certificate.

   ```
   -----BEGIN CERTIFICATE-----
   <PUPPET CA CERTIFICATE>
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
   <OPTIONAL INTERMEDIATE CA CERTIFICATES>
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
   <ROOT CA CERTIFICATE>
   -----END CERTIFICATE-----
   ```

5. Concatenate all of the CRLs into a PEM-encoded CRL chain, starting with any optional intermediate CA CRLs and ending with your root certificate CRL.

   ```
   -----BEGIN X509 CRL-----
   <OPTIONAL INTERMEDIATE CA CRLs>
   -----END X509 CRL-----
   -----BEGIN X509 CRL-----
   <ROOT CA CRL>
   -----END X509 CRL-----
   ```

   **Tip:** The PE installer automatically generates a Puppet CRL during installation, so you don't have to manage the Puppet CRL manually.

6. Copy your CA bundle (from step 4 on page 837), CRL chain (from step 5 on page 837), and private key (from step 2 on page 837) to the node where you're installing the primary server.

   **Tip:** Allow access to your private key only from the PE installation process, which runs as root.

7. Install PE using a customized pe.conf file with signing_ca parameters: ./puppet-enterprise-installer -c <PATH_TO_pe.conf>

   In your customized pe.conf file, you must include three keys for the signing_ca parameter: bundle, crl_chain, and private_key.

   ```
   {
   "pe_install::signing_ca": {
   "bundle": "/root/ca/int_ca_bundle"
   "crl_chain": "/root/ca/int_crl_chain"
   "private_key": "/root/ca/int_key"
   }
   }
   ```
8. Optional: Validate that the CA is working by running `puppet agent -t` and verifying your intermediate CA with OpenSSL.

```
openssl x509 -in /etc/puppetlabs/puppet/ssl/ca/signed/<HOSTNAME>.crt -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=intermediate-ca
...
```

**Use a custom SSL certificate for the console**

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.

**Before you begin**

- You need a X.509 cert, signed by the custom party CA, in PEM format, with matching private and public keys.
- If your custom cert is issued by an intermediate CA, the CA bundle must contain a complete chain, including the applicable root CA.
- These keys and certs must be in PEM format.
1. Retrieve the custom certificate and private key.

2. Move the certificate to /etc/puppetlabs/puppet/ssl/certs/console-cert.pem, replacing any existing file named console-cert.pem.

3. Move the private key to /etc/puppetlabs/puppet/ssl/private_keys/console-cert.pem, replacing any existing file named console-cert.pem.

4. If you previously specified a custom SSL certificate, remove any browser_ssl_cert and browser_ssl_private_key parameters specified in the console or Hiera data.

   In the console, go to the Node groups page, expand the PE Infrastructure group, and select the PE Console node group. Then, on the Classes tab, locate the puppet_enterprise::profile::console class, remove any browser_ssl_cert and browser_ssl_private_key parameters, and commit changes.

   If you declared these parameters on the Configuration data tab, remove them from that tab and commit changes.

   If you declared these parameters with Hiera, remove them from your Hiera data. For more information, refer to Configure settings with Hiera on page 287.

5. Run Puppet: puppet agent -t

You can navigate to your console and see the custom certificate in your browser.
Generate a custom Diffie-Hellman parameter file

The "Logjam Attack" (CVE-2015-4000) exposed several weaknesses in the Diffie-Hellman (DH) key exchange. To help mitigate the "Logjam Attack," PE ships with a pre-generated 2048 bit Diffie-Hellman param file. In the case that you don’t want to use the default DH param file, you can generate your own.

**Note:** In the following procedure, `<PROXY-CUSTOM-dhparam>.pem` can be replaced with any file name, except `dhparam_puppetproxy.pem`, as this is the default file name used by PE.

1. On your primary server, run:

   ```bash
   /opt/puppetlabs/puppet/bin/openssl dhparam -out /etc/puppetlabs/nginx/<PROXY-CUSTOM-dhparam>.pem 2048
   ```

   **Note:** After running this command, PE can take several minutes to complete this step.

2. Open your `pe.conf` file (located at `/etc/puppetlabs/enterprise/conf.d/pe.conf`) and add the following parameter and value:

   ```yaml
   "puppet_enterprise::profile::console::proxy::dhparam_file": "/etc/puppetlabs/nginx/<PROXY-CUSTOM-dhparam>.pem"
   ```

3. Run Puppet: `puppet agent -t`.

Enable TLSv1

TLSv1 and TLSv1.1 are disabled by default in PE.

You must enable TLSv1 to install agents on these platforms:

- AIX
- CentOS 5
- RHEL 5
- SLES 11
- Solaris 10, 11
- Windows Server 2008 R2

To comply with security regulations, PE 2019.1 and later uses only version 1.2 of the Transport Layer Security (TLS) protocol.

**CAUTION:** For nodes that use TLSv1, using a script to install or upgrade agents can fail if the curl version installed on the node uses OpenSSL earlier than version 1.0. This issue produces an SSL error during any curl connection to the primary server. As a workaround, add `--ciphers AES256-SHA` to `.curlrc` so that curl calls always use an appropriate cipher.

1. In the console, click **Node groups > PE Infrastructure**.
2. On the **Configuration data** tab, add the following class, parameter, and value:

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet_enterprise::master::puppetserver</td>
<td>ssl_protocols</td>
<td>[&quot;TLSv1&quot;, &quot;TLSv1.1&quot;, &quot;TLSv1.2&quot;]</td>
</tr>
</tbody>
</table>

3. Click **Add data**, and commit changes.
4. Run Puppet on the primary server and any compilers.
Maintenance

- **Backing up and restoring PE** on page 841
  Keep regular backups of your Puppet Enterprise (PE) infrastructure. Backups allow you to more easily migrate to new primary server hardware, migrate your PE installation after a major OS upgrade, troubleshoot your installation, and quickly recover from system failures.
- **Database maintenance** on page 845
  You can optimize the Puppet Enterprise (PE) databases to improve performance.

### Backing up and restoring PE

Keep regular backups of your Puppet Enterprise (PE) infrastructure. Backups allow you to more easily migrate to new primary server hardware, migrate your PE installation after a major OS upgrade, troubleshoot your installation, and quickly recover from system failures.

**Note:** These instructions are intended only for standard installations without disaster recovery.

**Important:** Store your backups in a secure, centralized location that you can access if your primary server fails.

You can use the `puppet-backup` command to back up and restore your primary server. You can't use this command to back up compilers. By default, the backup command creates a backup of:

- Your PE configuration, including license, classification, and RBAC settings. However, configuration backup does not include Puppet gems or Puppet Server gems.
- PE CA certificates and the full SSL directory.
- The Puppet code deployed to your code directory at backup time.
- PuppetDB data, including facts, catalogs and historical reports.

Each time you create a new backup, PE creates a single, timestamped backup file, named in the format `pe_backup-<TIMESTAMP>.tgz`. This file includes everything you're backing up. By default, PE writes backup files to `/var/puppetlabs/backups`, but you can change this location when you run the backup command. When you restore, specify the backup file you want to restore from.

If you are restoring to a previously existing primary server, uninstall and reinstall PE first. If you are restoring or migrating to a primary server with a different hostname, you can redirect your agents to the new primary server during the restore process. In both cases, you must restore to the same PE version used to create your backups. If your installation includes disaster recovery, you must complete the restore process before reprovisioning a new replica.

**Customizing backup and restore scope**

By default, backup and restore functions include your Puppet configuration, certificates, code, and PuppetDB. However, you can limit the scope of backup and restore with command line options. This allows you to back up to or restore from multiple files. This is useful if you want to back up some parts of your infrastructure more often than others.

For example, if you have frequent code changes, you might back up the code more often than you back up the rest of your infrastructure. When you limit backup scope, the backup file contains only the specified parts of your infrastructure. Be sure to give your backup file a name that identifies the scope so that you always know what a given file contains.

During restore, you must restore all scopes: code, configuration, certificates, and PuppetDB. However, you can restore each scope from different files, either by restoring from backup files with limited scope or by limiting the scope of the restore. For example, by specifying scope when you run the restore command, you could restore code, configuration, and certificates from one backup file and PuppetDB from a different one.
Back up your infrastructure

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take up to several hours depending on the size of PuppetDB.

**Before you begin**

To optionally encrypt your backup, you must import your GPG public key to your primary server.

Note: Ensure pg_repack won’t run prior to or during the backup/restore process.

By default, PE backs up files to /var/puppetlabs/backups.

1. On your primary server, run `sudo puppet-backup create --dir=<BACKUP_DIRECTORY>`, specifying a separate disk as your backup directory.

   You can specify these optional parameters:

   - `--name=<BACKUP_NAME>` – Specifies a name for the backup file. Default: `pe_backup-<TIMESTAMP>.tgz`
   - `--pe-environment=<ENVIRONMENT>` – Specifies an environment to back up. To ensure configuration is recovered correctly, this must be the environment where your primary server is located. Default: `production`
   - `--scope=<SCOPE_LIST>` – Specifies the data to back up: `certs`, `code`, `config`, `puppetdb`. Default: `all`
   - `--gpgkey=<KEY_ID>` – Specifies a GPG key ID to use for encrypting the backup file.
   - `--force` – Bypasses validation checks and ignore warnings.

2. Back up the secret keys directory located at `/etc/puppetlabs/orchestration-services/conf.d/secrets/`

   Secure the keys as you would any sensitive information.

   **CAUTION:** The backup command does not include the secret keys. You must back up this information separately.

Restore your infrastructure

Use the restore commands to migrate your primary server to a new host or to recover from system failure.

**Before you begin**

To decrypt an encrypted backup, you must import the GPG key pair (public and private keys) used for encryption to your new primary server.

**Note:** Ensure pg_repack won’t run prior to or during the backup/restore process.

1. If you are restoring to an existing primary server, purge any existing PE installation.
   a) On your primary server, uninstall PE by running `sudo /opt/puppetlabs/bin/puppet-enterprise-uninstaller -p -d`.
   b) Ensure that the directories `/opt/puppetlabs/` and `/etc/puppetlabs/` are no longer present on the system.

2. Install PE on the primary server you are restoring to. This must be the same PE version used to create the backup files.
   a) If you don’t have the PE installer script on the machine you want to restore to, download the installer tarball to the machine from the [download site](#), and unpack it by running `tar -xf <TARBALL_FILENAME>`.
   b) Navigate to the directory containing the install script. The installer script is located in the PE directory created when you unpacked the tarball.
   c) Install PE by running `sudo ./puppet-enterprise-installer`
3. On your primary server, run

```bash
sudo puppet-backup restore <BACKUP-FILENAME>
```

to restore your PE infrastructure.

You can specify these optional parameters:

- `--pe-environment=<ENVIRONMENT>` – Specifies an environment to restore. Default: production
- `--scope=<SCOPE_LIST>` – Specifies the data to restore: certs, code, config, puppetdb. Default: all
- `--force` – Bypasses validation checks and ignore warnings.

4. Restore your backup of the secret keys directory used to encrypt and decrypt sensitive data stored in the inventory service.

The secret keys directory is located at:

```
/etc/puppetlabs/orchestration-services/conf.d/secrets/
```

**CAUTION:** The restore command does not include the secret keys. You must restore this information separately.

5. Ensure the secret key ownership is `pe-orchestration-services:pe-orchestration-services`.

6. Restart the `pe-orchestration-services` service.

7. If you restored PE onto a primary server with a different hostname than the original installation, and you have not configured the `dns_alt_names` setting in the `pe.conf` file, redirect your agents to the new primary server. An easy way to do this is by running a task with the Bolt task runner.

   a) Download and install Bolt, if you don't already have it installed.
   
   b) Update the `puppet.conf` file to point all agents at the new primary server by running:

   ```
bolt task run puppet_conf action=set section=agent setting=server value=<RESTORE_HOSTNAME> --nodes <COMMA-SEPARATED LIST OF NODES>
bolt task run puppet_conf action=set section=main setting=server value=<RESTORE_HOSTNAME> --nodes <COMMA-SEPARATED LIST OF NODES>
   ```

   c) Run `puppet agent -t --no-use_cached_catalog` on the newly restored primary server **twice** to apply changes and then restart services.

   d) Run `puppet agent -t --no-use_cached_catalog` on all agent nodes to test connection to the new primary server.

8. If your infrastructure had Code Manager enabled when your backup file was created, deploy your code by running:

   ```
puppet-access login
puppet code deploy --all --wait
   ```

9. If your installation includes disaster recovery, reprovision a new replica.

**Related information**

**Uninstalling** on page 254

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

**Installing PE** on page 209

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

**Provision and enable a replica** on page 325
Provisioning a replica duplicates specific components and services from the primary server to the replica. Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.

**Directories and data backed up**

These are the directories and data included in PE backups.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Directories and databases backed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>certs</td>
<td>/etc/puppetlabs/puppet/ssl/</td>
</tr>
<tr>
<td>code</td>
<td>/etc/puppetlabs/code/</td>
</tr>
<tr>
<td></td>
<td>/etc/puppetlabs/code-staging/</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/</td>
</tr>
<tr>
<td></td>
<td>puppetserver/filesync/storage/</td>
</tr>
<tr>
<td>config</td>
<td>Orchestrator database</td>
</tr>
<tr>
<td></td>
<td>RBAC database</td>
</tr>
<tr>
<td></td>
<td>Classifier database</td>
</tr>
<tr>
<td></td>
<td>/etc/puppetlabs/. except:</td>
</tr>
<tr>
<td></td>
<td>/etc/puppetlabs/code/</td>
</tr>
<tr>
<td></td>
<td>/etc/puppetlabs/code-staging/</td>
</tr>
<tr>
<td></td>
<td>/etc/puppetlabs/puppet/ssl</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/. except:</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/puppet</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/pe_build</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/packages</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/apps</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/postgresql</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/enterprise/modules</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/puppetserver/vendored-jruby-gems</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/bin</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/client-tools</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/share</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/puppetserver/filesync/storage</td>
</tr>
<tr>
<td></td>
<td>/opt/puppetlabs/server/data/puppetserver/filesync/client</td>
</tr>
</tbody>
</table>
### Database maintenance

You can optimize the Puppet Enterprise (PE) databases to improve performance.

#### Enable the `pe_databases` module

The `pe_databases` module helps you manage and tune your PE databases. The module is installed in the `$basemodulepath` directory as part of the PE installation or upgrade process, and it is enabled by default.

**Important:** If you have a version of this module, from the Forge or other sources, specified in the code, you must remove this version before upgrading to allow the version bundled with PE to be asserted.

1. To enable or disable the `pe_databases` module, change the `puppet_enterprise::enable_database_maintenance` parameter. This parameter accepts Boolean values.
2. Run Puppet: `puppet agent -t`

#### Databases in Puppet Enterprise

PE uses PostgreSQL as the backend for its databases. Use the native tools in PostgreSQL to perform database exports and imports.

The PE PostgreSQL database includes the following databases:

<table>
<thead>
<tr>
<th>Database</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-activity</td>
<td>Activity data from the Classifier, including user, nodes, and time of activity.</td>
</tr>
<tr>
<td>pe-classifier</td>
<td>Classification data, all node group information.</td>
</tr>
<tr>
<td>pe-puppetdb</td>
<td>PuppetDB data, including exported resources, catalogs, facts, and reports.</td>
</tr>
<tr>
<td>pe-rbac</td>
<td>Role-based access control data, including users, permissions, and AD/LDAP info.</td>
</tr>
<tr>
<td>pe-orchestrator</td>
<td>Orchestrator data, including user, node, and result details about job runs.</td>
</tr>
</tbody>
</table>

#### List all database names

Use these instructions to list all database names.

To generate a list of database names:

1. **Assume the `pe-postgres` user:**

   ```bash
   sudo su - pe-postgres -s /bin/bash
   ```
2. Open the PostgreSQL command-line:

```
/opt/puppetlabs/server/bin/psql
```

3. List the databases:

```
l
```

4. Exit the PostgreSQL command line:

```
q
```

5. Log out of the pe-postgres user:

```
logout
```

---

### Troubleshooting

Use this guide to troubleshoot issues with your Puppet Enterprise installation.

**Important:** Before you troubleshoot, view the What gets installed and where? on page 202 page. PE installs several software components, configuration files, databases, services and users, and log files. It is useful to know their locations when you need to troubleshoot your infrastructure.

- **Log locations** on page 847
  The software distributed with PE generates log files you can use for troubleshooting.
- **Troubleshooting installation** on page 848
  If installation fails, check for these issues.
- **Troubleshooting disaster recovery** on page 848
  If disaster recovery commands fail, check for these issues.
- **Troubleshooting puppet infrastructure run commands** on page 849
  If puppet infrastructure run commands fail, review the logs at `/var/log/puppetlabs/installer/bolt_info.log` and check for these issues.
- **Troubleshooting connections between components** on page 850
  If agent nodes can't retrieve configurations, check for communication, certificate, DNS, and NTP issues.
- **Troubleshooting the databases** on page 851
  If you have issues with the databases that support the console, make sure that the PostgreSQL database is not too large or using too much memory, that you don't have port conflicts, and that puppet apply is configured correctly.
- **Troubleshooting cloud deployments** on page 852
  If you encounter issues with your PE cloud deployment, try these fixes.
- **Troubleshooting Code Manager**
• **Troubleshooting Windows** on page 853
  Troubleshoot Windows issues with failed installations and upgrades, and failed or incorrectly applied manifests. Use the error message reference to solve your issues. Enable debugging.
• **Troubleshooting backup and restore** on page 857
  If backup or restore fails, check for these issues.

## Log locations

The software distributed with PE generates log files you can use for troubleshooting.

### Primary server logs

- /var/log/puppetlabs/puppetserver/code-manager-access.log
- /var/log/puppetlabs/puppetserver/file-sync-access.log
- /var/log/puppetlabs/puppetserver/pcp-broker.log — The log file for Puppet Communications Protocol brokers on compilers.
- /var/log/puppetlabs/puppetserver/puppetserver.log — The primary server logs its activity, including compilation errors and deprecation warnings, here.
- /var/log/puppetlabs/puppetserver/puppetserver-access.log
- /var/log/puppetlabs/puppetserver/puppetserver-daemon.log — This is where fatal errors or crash reports can be found.
- /var/log/puppetlabs/puppetserver/puppetserver-status.log

### Agent logs

The locations of agent logs depend on the agent operating system.

On *nix nodes, the agent service logs its activity to the syslog service. Your syslog configuration dictates where these messages are saved, but the default location is /var/log/messages on Linux, /var/log/system.log on macOS, and /var/adm/messages on Solaris.

On Windows nodes, the agent service logs its activity to the event log. You can view its logs by browsing the event viewer.

### Console and console services logs

- /var/log/puppetlabs/console-services/console-services.log
- /var/log/puppetlabs/console-services/console-services-api-access.log
- /var/log/puppetlabs/console-services-console-services-access.log
- /var/log/puppetlabs/console-services-console-services-daemon.log — This is where fatal errors or crash reports can be found.
- /var/log/puppetlabs/nginx/access.log
- /var/log/puppetlabs/nginx/error.log — Contains errors related to nginx. Console errors that aren’t logged elsewhere can be found in this log.

### Installer logs

- /var/log/puppetlabs/installer/http.log — Contains web requests sent to the installer. This log is present only on the machine from which a web-based installation was performed.
- /var/log/puppetlabs/installer/orchestrator_info.log — Contains run details about puppet infrastructure commands that use the orchestrator, including the commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.
- /var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log — Contains the contents of the last installer run.
/var/log/puppetlabs/installer/installer-<timestamp>.log — Contains the operations performed and any errors that occurred during installation.

/var/log/puppetlabs/installer/<action_name>_<timestamp>_<run_description>.log — Contains details about disaster recovery command execution. Each action triggers multiple Puppet runs, some on the primary server, some on the replica, so there might be multiple log files for each command.

**Database logs**

- /var/log/puppetlabs/postgresql/11.15/pgstartup.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Mon.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Tue.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Wed.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Thu.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Fri.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Sat.log
- /var/log/puppetlabs/postgresql/11.15/postgresql-Sun.log
- /var/log/puppetlabs/puppetdb/puppetdb.log
- /var/log/puppetlabs/puppetdb/puppetdb-access.log
- /var/log/puppetlabs/puppetdb/puppetdb-status.log

**Troubleshooting installation**

If installation fails, check for these issues.

*Note:* If you encounter errors during installation, you can troubleshoot and run the installer as many times as needed.

**DNS is misconfigured**

DNS must be configured correctly for successful installation.

1. Verify that agents can reach the primary server hostname you chose during installation.
2. Verify that the primary server can reach itself at the primary server hostname you chose during installation.
3. If the primary server and console components are on different servers, verify that they can communicate with each other.

**Security settings are misconfigured**

Firewall and security settings must be configured correctly for successful installation.

1. On your primary server, verify that inbound traffic is allowed on ports 8140 and 443.
2. If your primary server has multiple network interfaces, verify that the primary server allows traffic via the IP address that its valid DNS names resolve to, not just via an internal interface.

**Troubleshooting disaster recovery**

If disaster recovery commands fail, check for these issues.

**Latency over WAN**

If the primary server and replica communicate over a slow, high latency, or lossy connection, the provision and enable commands can fail.

If this happens, try re-running the command.
Replica is connected to a compiler instead of a primary server

The provision command triggers an error if you try to provision a replica node that’s connected to a compiler. The error is similar to the following:

```
Failure during provision command during the puppet agent run on replica 2:
Failed to generate additional resources using 'eval_generate':
Error 500 on SERVER: Server Error: Not authorized
to call search on /file_metadata/pe_modules with
{:rest=>"pe_modules", :links=>"manage", :recurse=>true, :source_permissions=>"ignore",
Source: /Stage[main]/Puppet_enterprise::Profile::Primary_master_replica/
File[/opt/puppetlabs/server/share/installer/modules]File: /opt/
puppetlabs/puppet/modules/puppet_enterprise/manifests/profile/
primary_master_replica.ppLine: 64
```

On the replica you want to provision, edit /etc/puppetlabs/puppet.conf so that the server and server_list settings use a primary server, rather than a compiler.

Both server and server_list are set in the agent configuration file

When the agent configuration file contains settings for both server and server_list, a warning appears. This warning can occur after enabling a replica. You can ignore the warning, or hide it by removing the server setting from the agent configuration, leaving only server_list.

Node groups are empty

When provisioning and enabling a replica, the orchestrator is used to run Puppet on different groups of nodes. If a group of nodes is empty, the tool reports that there's nothing for it to do and the job is marked as failed in the output of puppet job show. This is expected, and doesn't indicate a problem.

**Troubleshooting puppet infrastructure run commands**

If puppet infrastructure run commands fail, review the logs at /var/log/puppetlabs/installer/bolt_info.log and check for these issues.

Running commands when logged in as a non-root user

All puppet infrastructure run commands require you to act as the root user on all nodes that the command touches. If you are trying to run a puppet infrastructure run command as a non-root user, you must be able to SSH to the impacted nodes (as the same non-root user) in order for the command to succeed.

When you run a puppet infrastructure run command, Bolt uses your system's existing OpenSSH ssh_config configuration file to connect to your nodes. If this file is missing or configured incorrectly, Bolt tries to connect as root. To make sure the correct user connects to the nodes, you have the following options.

1. Set up your OpenSSH ssh_config configuration file to point to a user with sudo privileges:

   ```
   Host *.example.net
   UserKnownHostsFile=~/.ssh/known_hosts
   User <USER WITH SUDO PRIVILEGES>
   ```

2. When running a puppet infrastructure run command, pass in the --user <USER WITH SUDO PRIVILEGES> flag.

   If your sudo configuration requires a password to run commands, pass in the --sudo-password <PASSWORD> flag when running a puppet infrastructure run command.

   **Note:** To avoid logging your password to .bash_history, set HISTCONTROL=ignorebespace in your .bashrc file and add a space to the beginning of the command.
If your operating system distribution includes the `requiretty` option in its `/etc/sudoers` file, you must either remove this option from the file or pass the `--tty` flag when running a `puppet infrastructure run` command.

### Passing hashes from the command line

When passing a hash on the command line as part of a `puppet infrastructure run` command, the hash must be quoted, much like a JSON object. For example:

```
'{"parameter_one": "value_one", "parameter_two": "value_two"}"
```

### Troubleshooting connections between components

If agent nodes can't retrieve configurations, check for communication, certificate, DNS, and NTP issues.

#### Agents can't reach the primary server

Agent nodes must be able to communicate with the primary server in order to retrieve configurations.

If agents can't reach the primary server, running `telnet <PRIMARY_HOSTNAME> 8140` returns the error "Name or service not known."

1. Verify that the primary server is reachable at a DNS name your agents recognize.
2. Verify that the `pe-puppetserver` service is running.

#### Agents don't have signed certificates

Agent certificates must be signed by the primary server.

If the node's Puppet agent logs have a warning about unverified peer certificates in the current SSL session, the agent has submitted a certificate signing request that hasn't yet been signed.

1. On the primary server, view a list of pending certificate requests: `puppet cert list`
2. Sign a specified node's certificate: `puppetserver ca sign <NODE_NAME>`

#### Agents aren't using the primary server's valid DNS name

Agents trust the primary server only if they contact it at one of the valid hostnames specified when the primary server was installed.

On the node, if the results of `puppet agent --configprint server` don't return one of the valid DNS names you chose during installation of the primary server, the node and primary server can't establish communication.

1. To edit the primary server's hostname on nodes, in `/etc/puppetlabs/puppet/puppet.conf`, change the `server` setting to a valid DNS name.
2. To reset the primary server's valid DNS names, log in as root and run:

```
puppet infrastructure run regenerate_primary_certificate --dns_alt_names=<COMMA-SEPARATED_LIST_OF_DNS_NAMES>
```

#### Time is out of sync

The date and time must be in sync on the primary server's and agent nodes.

If time is out of sync on nodes, running `date` returns incorrect or inconsistent dates.

Get time in sync by setting up NTP. Keep in mind that NTP can behave unreliably on virtual machines.
**Node certificates have invalid dates**

The date and time must be in sync when certificates are created.

If certificates were signed out of sync, running `openssl x509 -text -noout -in $(puppet config print --section master ssldir)/certs/<NODE NAME>.pem` returns invalid dates, such as certificates dated in the future.

1. On the primary server, delete certificates with invalid dates: `puppetserver ca clean --certname <NODE CERT NAME>`
2. On nodes with invalid certificates, delete the SSL directory: `rm -r $(puppet config print --section master ssldir)`
3. On agent nodes, generate a new certificate request: `puppet agent --test`
4. On the primary server, sign the request: `puppetserver ca sign <NODE NAME>`

**A node is re-using a certname**

If a node re-uses an old node's certname and the primary server retains the previous node's certificate, the new node is unable to request a new certificate.

1. On the primary server, clear the node's certificate: `puppetserver ca clean --certname <NODE CERT NAME>`
2. On agent node, generate a new certificate request: `puppet agent --test`
3. On the primary server, sign the request: `puppetserver ca sign <NODE NAME>`

**Agents can't reach the filebucket server**

If the primary server is installed with a certname that doesn't match its hostname, agents can't back up files to the filebucket on the primary server.

If agents log errors like "could not back up," nodes are likely attempting to back up files to the wrong hostname.

On the primary server, edit `/etc/puppetlabs/code/environments/production/manifests/site.pp` so that filebucket `server` attribute points to the correct hostname:

```plaintext
# Define filebucket 'main':
filebucket { 'main':
  server => '<PRIMARY_DNS_NAME>',
  path   => false,
}
```

Changing the filebucket server attribute on the primary server fixes the error on all agent nodes.

**Orchestrator can't connect to PE Bolt server**

Debug a faulty connection between the orchestrator and PE Bolt server by setting the `bolt_server_loglevel` in the `puppet_enterprise::profile::bolt_server` class and running Puppet, or by manually updating `loglevel` in `/etc/puppetlabs/bolt-server/conf.d/bolt-server.conf`. The server logs are located at `/var/log/puppetlabs/bolt-server/bolt-server.log`.

**Troubleshooting the databases**

If you have issues with the databases that support the console, make sure that the PostgreSQL database is not too large or using too much memory, that you don't have port conflicts, and that `puppet apply` is configured correctly.

**Note:** If you're using your own instance of PostgreSQL for the console and PuppetDB, you must use version 9.1 or higher.

© 2022 Puppet, Inc., a Perforce company
PostgreSQL is taking up too much space

The PostgreSQL autovacuum=on setting prevents the database from growing too large and unwieldy. Routine vacuuming is turned on by default.

Verify that autovacuum is set to on.

PostgreSQL buffer memory causes installation to fail

When installing PE on machines with large amounts of RAM, the PostgreSQL database might use more shared buffer memory than is available.

If this issue is present, /var/log/pe-postgresql/pgstartup.log shows the error:

```
FATAL: could not create shared memory segment: No space left on device
DETAIL: Failed system call was shmget(key=5432001, size=34427584512,03600).
```

1. On the primary server, set the shmmax kernel setting to approximately 50% of the total RAM.
2. Set the shmall kernel setting to the quotient of the new shmmax setting divided by the page size. You can confirm page size by running `getconf PAGE_SIZE`.
3. Set the new kernel settings:

```
sysctl -w kernel.shmmax=<your shmmax calculation>
sysctl -w kernel.shmall=<your shmall calculation>
```

The default port for PuppetDB conflicts with another service

By default, PuppetDB communicates over port 8081. In some cases, this might conflict with existing services, for example McAfee ePolicy Orchestrator.

Install in text mode with a parameter in `pe.conf` that specifies a different port using `puppet_enterprise::puppetdb_port`.

puppet resource generates Ruby errors after connecting puppet apply to PuppetDB

If Puppet apply is configured incorrectly, for example by modifying `puppet.conf` to add the parameters `storeconfigs_backend = puppetdb` and `storeconfigs = true` in both the main and primary server sections, `puppet resource` ceases to function and displays a Ruby run error.

Modify `/etc/puppetlabs/puppet/routes.yaml` to correctly connect Puppet apply without affecting other functions.

Troubleshooting cloud deployments

If you encounter issues with your PE cloud deployment, try these fixes.

Related information

Cloud PE on page 17
Puppet Enterprise 2019.8 cloud images are available from several providers.

Authentication fails with SSH username or credentials

Cloud providers vary in their support for SSH authentication.

AWS
AWS cloud deployments use cloud-init to provision an SSH key for the puppetadmin user, but disable root SSH access. You must specify a key pair when launching an AWS Marketplace image and connect it with the matching private key. For example, you can run:

```bash
aws ec2 run-instance --key-name <KEYPAIR_NAME> ...
...
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

**Azure**

Azure cloud deployments support authenticating with either an SSH key pair or a user name and password. The private key must be the pair of the public key specified when you created the image.

**After 60 days, the puppetadmin user account stops working**

The default puppetadmin user’s password expires 60 days after the image is created. If you fail to reset the password, the account expires.

To prevent the password from expiring, run `chage -E -1 puppetadmin` on the primary server.

**Agent run fails for non-root users**

An agent run initiated by puppetadmin or any other non-root user fails when attempting to access certificates, packages, and services.

Always execute agent runs with super-user privileges:

```bash
sudo /usr/local/bin/puppet agent -t
```

**Certificate-signing curl command has incorrect URL**

The curl command on the console's Unsigned Certificates page contains a URL that uses your primary server's private hostname or internal DNS name, which won't work for nodes that can't resolve that name.

For example, the command might look like:

```bash
curl -k https://puppetmasterv2.liweionmsdnwoe.xx.internal.cloudapp.net:8140/packages/current/install.bash | sudo bash
```

Change the private hostname or internal DNS name to the public hostname or externally qualified domain name.

---

**Troubleshooting Windows**

Troubleshoot Windows issues with failed installations and upgrades, and failed or incorrectly applied manifests. Use the error message reference to solve your issues. Enable debugging.

**Installation fails**

Check for these issues if Windows installation with Puppet fails.

**The installation package isn't accessible**

The source of an MSI or EXE package must be a file on either a local filesystem, a network mapped drive, or a UNC path.

RI-based installation sources aren't supported, but you can achieve a similar result by defining a file whose source is the primary server and then defining a package whose source is the local file.
Installation wasn’t attempted with admin privileges

Puppet fails to install when trying to perform an unattended installation from the command line. A "norestart" message is returned, and installation logs indicate that installation is forbidden by system policy.

You must install as an administrator.

Upgrade fails

The Puppet MSI package overwrites existing entries in the puppet.conf file. If you upgrade or reinstall using a different primary server hostname, Puppet applies the new value in $confdir\puppet.conf.

When you upgrade Windows, you must use the same primary server hostname that you specified when you installed.

For information on what settings are preserved during an upgrade, see installing Install Windows agents on page 231.

Errors when applying a manifest or doing a Puppet agent run

Check for the following issues if manifests cannot be applied or are not applied correctly on Windows nodes.

Path or file separators are incorrect

For Windows, path separators must use a semi-colon (;), while file separators must use forward or backslashes as appropriate to the attribute.

In most resource attributes, the Puppet language accepts either forward or backslashes as the file separator. However, some attributes absolutely require forward slashes, and some attributes absolutely require backslashes.

When backslashes are double-quoted(""), they must be escaped. When single-quoted (''), they can be escaped. For example, these are valid file resources:

```puppet
file { 'c:\path\to\file.txt': }
file { 'c:\path\\to\\file.txt': }
file { "c:\path\to\file.txt": }
```

But this is an invalid path, because \p, \t, and \f are interpreted as escape sequences:

```puppet
file { "c:\path\to\file.txt": }
```

For more information, see the language reference about backslashes on Windows.

Cases are inconsistent

Several resources are case-insensitive on Windows, like files, users, groups. However, these resources can be case sensitive in Puppet.

When establishing dependencies among resources, make sure to specify the case consistently. Otherwise, Puppet can't resolve dependencies correctly. For example, applying this manifest fails, because Puppet doesn't recognize that ALEX and alex are the same user:

```puppet
file { 'c:\foo\bar':
  ensure => directory,
  owner  => 'ALEX'
}
user { 'alex':
  ensure => present
}
...
err: /Stage[main]/File[c:\foo\bar]: Could not evaluate: Could not find user ALEX
```
Shell built-ins are not executed

Puppet doesn't support a shell provider on Windows, so executing shell built-ins directly fails.

Wrap the built-in in `cmd.exe`:

```ruby
exec { 'cmd.exe /c echo foo':
  path => 'c:\windows\system32;c:\windows'
}
```

Tip: In the 32-bit versions of Puppet, you might encounter file system redirection, where `system32` is switched to `sysWoW64` automatically.

PowerShell scripts are not executed

By default, PowerShell enforces a restricted execution policy which prevents the execution of scripts.

Specify the appropriate execution policy in the PowerShell command, for example:

```ruby
exec { 'test':
  command => 'powershell.exe -executionpolicy remotesigned -file C:\test.ps1',
  path    => $::path
}
```

Or use the Puppet supported PowerShell.

Services are referenced with display names instead of short names

Windows services support a short name and a display name, but Puppet uses only short names.

1. Verify that your Puppet manifests use short names, for example `wuauserv`, not `Automatic Updates`.

Error messages

Use this reference to troubleshoot error messages when using Windows with Puppet.

- **Error:** Could not connect via HTTPS to `https://forge.puppet.com` / Unable to verify the SSL certificate / The certificate may not be signed by a valid CA / The CA bundle included with OpenSSL may not be valid or up to date

  This error occurs when you run the `puppet module` subcommand on newly provisioned Windows nodes. The Forge uses an SSL certificate signed by the GeoTrust Global CA certificate. Newly provisioned Windows nodes might not have that CA in their root CA store yet.

  Download the "GeoTrust Global CA" certificate from GeoTrust's list of root certificates and manually install it by running `certutil -addstore Root GeoTrust_Global_CA.pem`.

- **Service 'Puppet Agent' (puppet) failed to start.** Verify that you have sufficient privileges to start system services.

  This error occurs when installing Puppet on a UAC system from a non-elevated account. Although the installer displays the UAC prompt to install Puppet, it does not elevate privileges when trying to start the service.

  Run from an elevated `cmd.exe` process when installing the MSI.

- **Cannot run on Microsoft Windows without the sys-admin, win32-process, win32-dir, win32-service and win32-taskscheduler gems.**

  This error occurs if you attempt to run Windows without required gems.

  Install specified gems: `gem install <GEM_NAME>`
• err: /Stage[main]//Scheduled_task[task_system]: Could not evaluate: The operation completed successfully.

This error occurs when using a the task scheduler gem earlier than version 0.2.1.

Update the task scheduling gem: gem update win32-taskscheduler

• err: /Stage[main]//Exec[C:/tmp/foo.exe]/returns: change from notrun to 0 failed: CreateProcess() failed: Access is denied.

This error occurs when requesting an executable from a remote primary server that cannot be executed.

Set the user/group executable bits appropriately on the primary server:

```ruby
file { "C:/tmp/foo.exe":
  source => "puppet:///modules/foo/foo.exe",
}
exec { 'C:/tmp/foo.exe':
  logoutput => true
}
```

• err: getaddrinfo: The storage control blocks were destroyed.

This error occurs when the agent can’t resolve a DNS name into an IP address or if the reverse DNS entry for the agent is wrong.

Verify that you can run `nslookup <dns>`. If this fails, there is a problem with the DNS settings on the Windows agent, for example, the primary dns suffix is not set. For more information, see Microsoft's documentation on Naming Hosts and Domains.

• err: Could not request certificate: The certificate retrieved from the master does not match the agent's private key.

This error occurs if the agent's SSL directory is deleted after it retrieves a certificate from the primary server, or when running the agent in two different security contexts.

Elevate privileges using Run as Administrator when you select Start Command Prompt with Puppet.

• err: Could not send report: SSL_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed. This is often because the time is out of sync on the server or client.

This error occurs when Windows agents' time isn’t synched. Windows agents that are part of an Active Directory domain automatically have their time synchronized with AD.

For agents that are not part of an AD domain, enable and add the Windows time service manually:

```bash
w32tm /register
net start w32time
w32tm /config /manualpeerlist:<ntpserver> /syncfromflags:manual /update
w32tm /resync
```

• Error: Could not parse for environment production: Syntax error at '='; expected '}'

This error occurs if puppet apply -e is used from the command line and the supplied command is surrounded with single quotes (‘), which causes cmd.exe to interpret any => in the command as a redirect.

Surround the command with double quotes (") instead.
Logging and debugging

The Windows Event Log can be helpful when troubleshooting issues with Windows.

Enable Puppet to emit \texttt{--debug} and \texttt{--trace} messages to the Windows Event Log by stopping the Puppet service and restarting it:

\begin{verbatim}
c:\>sc stop puppet && sc start puppet --debug --trace
\end{verbatim}

Note that this setting takes effect only until the next time the service is restarted, or until the system is rebooted.

Related information

Logging for Puppet agent on Windows systems
Log files installed on page 207

The software distributed with PE generates log files that you can collect for compliance or use for troubleshooting.

Troubleshooting backup and restore

If backup or restore fails, check for these issues.

The \texttt{puppet-backup create} command fails with the error \texttt{command puppet infrastructure recover_configuration failed}

The \texttt{puppet-backup create} command might fail if any gem installed on the Puppet Server isn't present on the agent environment on the primary server. If the gem is either absent or of a different version on the primary server's agent environment, you get the error "\texttt{command puppet infrastructure recover_configuration failed}".

To fix this, install missing or incorrectly versioned gems on the primary server's agent environment. To find which gems are causing the error, check the backup logs for gem incompatibility issues with the error message. PE creates backup logs as a \texttt{report.txt} whenever you run a \texttt{puppet-backup} command.

To see what gems and their versions you have installed on your Puppet Server, run the command \texttt{puppetserver gem list}. To see what gems are installed in the agent environment on your primary server, run \texttt{/opt/puppetlabs/puppet/bin/gem list}.

The \texttt{puppet-backup restore} command fails with errors about a duplicate operator family

When restoring the \texttt{pe-rbac} database, the restore process exits with errors about a duplicate operator family, \texttt{citext_ops}.

To work around this issue:

1. Log into your PostgreSQL instance:

\begin{verbatim}
sudo su - pe-postgres -s /bin/bash -c "/opt/puppetlabs/server/bin/psql pe-rbac"
\end{verbatim}

2. Run these commands:

\begin{verbatim}
ALTER EXTENSION citext ADD operator family citext_ops using btree;
ALTER EXTENSION citext ADD operator family citext_ops using hash;
\end{verbatim}

3. Exit the PostgreSQL shell and re-run the backup utility.