## Contents

### Getting started

- Getting started on *nix ................................................................. 74
- Start installing modules ............................................................ 79
- Start installing *nix agents ......................................................... 77
- Start installing *nix agents ......................................................... 79

### Release notes

- New features ................................................................................ 35
- Enhancements ............................................................................... 36
- Deprecations and removals ......................................................... 44
- Known issues ............................................................................... 45
  - Installation and upgrade known issues ................................... 46
  - High availability known issues ............................................... 48
  - PuppetDB and PostgreSQL known issues ............................ 49
  - Puppet Server known issues .................................................. 50
  - Puppet and Puppet services known issues ......................... 51
  - Supported platforms known issues ...................................... 52
  - Configuration and maintenance known issues .................... 53
  - Console and console services known issues ....................... 54
  - Orchestration services known issues .................................... 55
  - Orchestration services known issues .................................... 55
  - Permissions known issues ..................................................... 55
  - Code management known issues ......................................... 55
  - Razor known issues ............................................................. 57
  - Internationalization known issues ........................................ 57
- Resolved issues ........................................................................... 58
- What's new since PE 2016.4 ....................................................... 65

### Getting started on *nix

- PE architecture ........................................................................... 16
- The master of masters and compile masters ............................. 16
- The Puppet agent ................................................................. 17
- Console services ........................................................................ 18
- Code Manager and r10k ......................................................... 20
- Orchestration services ............................................................ 20
- PE databases ............................................................................. 21
- Security and communications ............................................... 21
- Component versions in recent PE releases .............................. 22
  - Master and agent compatibility ........................................... 26
  - Task compatibility .............................................................. 26
- PE and open source version numbers ...................................... 26
- Getting support .......................................................................... 27
  - Puppet Enterprise support life cycle ................................... 27
  - The customer support portal ............................................... 27
  - Getting support from the Puppet community ..................... 29
- API index ................................................................................ 30
- Puppet platform documentation for PE .................................. 31
- Translated PE ........................................................................... 33

### API index

- Component versions in recent PE releases .............................. 22
- Getting support .......................................................................... 27
- Getting support from the Puppet community ......................... 29
- API index ................................................................................ 30
- Puppet platform documentation for PE .................................. 31
- Translated PE ........................................................................... 33
Installing .................................................................................................................144
Choosing an architecture..........................................................................................145
System requirements................................................................................................148
Hardware requirements..........................................................................................148
Supported operating systems..................................................................................152
Supported browsers..................................................................................................157
System configuration...............................................................................................158
What gets installed and where?................................................................................163
Software components installed..............................................................................163
Executable binaries and symlinks installed............................................................167
Modules and plugins installed..................................................................................167
Configuration files installed....................................................................................167
Tools installed..........................................................................................................167
Databases installed.................................................................................................168
Services installed...................................................................................................168
User and group accounts installed..........................................................................169
Log files installed...................................................................................................169
Certificates installed...............................................................................................171
Installing Puppet Enterprise....................................................................................172
Download and verify the installation package.........................................................172
Install using web-based installation (mono configuration). ..................................173
Install using text mode (mono configuration). ......................................................174
Install using text mode (split configuration). .........................................................174
Web-based installation prerequisites....................................................................176
Web-based installation options..............................................................................177
Text mode installer options....................................................................................178
Configuration parameters and the pe.conf file.....................................................178
Purchasing and installing a license key....................................................................186
Getting licensed......................................................................................................186
Install a license key................................................................................................186
Verify installed licenses and active nodes..............................................................186
Installing agents.....................................................................................................186
Using the install script ............................................................................................187
Installing *nix agents.............................................................................................190
Installing Windows agents.................................................................192
Installing macOS agents......................................................................197
Installing non-root agents..............................................................198
Managing certificate signing requests..............................................201
Configuring agents........................................................................202
Installing network device agents..................................................202
Installing Arista EOS agents.........................................................202
Installing Cisco agents.................................................................204
Installing compile masters.............................................................205
  How compile masters work........................................................205
  Using load balancers with compile masters..............................206
  Install compile masters.............................................................207
  Configure compile masters.........................................................207
Installing ActiveMQ hubs and spokes...........................................209
  Setting up MCollective..............................................................209
  Install ActiveMQ hubs and spokes............................................209
Installing PE client tools...............................................................213
  Supported PE client tools operating systems............................214
  Install PE client tools on a managed workstation....................214
  Install PE client tools on an unmanaged workstation...............215
  Configuring and using PE client tools.......................................216
Installing external PostgreSQL.....................................................217
  Install standalone PE-PostgreSQL............................................218
  Install unmanaged PostgreSQL..................................................218
  External PostgreSQL options for web-based installation...........220
Uninstalling.......................................................................................221
  Uninstall component nodes......................................................221
  Uninstall agents..........................................................................221
  Uninstaller options......................................................................222

**Upgrading**......................................................................................223
  Upgrading Puppet Enterprise......................................................223
    Upgrade paths..............................................................................223
    Upgrade cautions........................................................................223
    Test modules before upgrade..................................................224
    Upgrade a monolithic installation..............................................225
    Upgrade with high availability enabled....................................226
    Upgrade a split or large environment installation....................227
    Migrate from a split to a monolithic installation......................228
    Upgrading PostgreSQL.............................................................229
    Text mode installer options......................................................230
    Checking for updates..................................................................230
  Upgrading agents........................................................................230
    Upgrade *nix or Windows agents using the puppet_agent module...231
    Upgrade a *nix or Windows agent using a script.......................232
    Upgrading the agent independent of PE....................................233
    Upgrade agents without internet access....................................234

**Configuring Puppet Enterprise**..................................................234
  Methods for configuring Puppet Enterprise..............................235
    Configure settings using the console.........................................236
    Configure settings with Hiera...................................................236
    Configure settings in pe.conf....................................................237
  Configuring and tuning Puppet Server........................................237
Configuring high availability........................................................................................................... 264
High availability.................................................................................................................................. 264
High availability architecture................................................................................................................. 264
What happens during failovers............................................................................................................... 266
System and software requirements.................................................................266
Classification changes in high availability installations...............................267
Load balancer timeout in high availability installations.................................268
Configure high availability.............................................................................269
Provision a replica..........................................................................................269
Enable a replica...............................................................................................270
Promote a replica............................................................................................272
Enable a new replica using a failed master.....................................................272
Forget a replica...............................................................................................273
Reinitialize a replica.......................................................................................273

Accessing the console.....................................................................................273
Reaching the console.......................................................................................274
Accepting the console’s certificate.................................................................274
Logging in.......................................................................................................274
Generate a user password reset token............................................................274
Reset the admin password.............................................................................275
Troubleshooting login to the PE admin account..........................................275

Managing access...........................................................................................275
User permissions and user roles.................................................................276
Structure of user permissions.........................................................................276
User permissions............................................................................................277
Working with node group permissions..........................................................280
Best practices for assigning permissions......................................................280
Creating and managing local users and user roles........................................281
Create a new user...........................................................................................281
Give a new user access to the console............................................................281
Create a new user role...................................................................................281
Assign permissions to a user role.................................................................282
Add a user to a user role................................................................................282
Remove a user from a user role......................................................................282
Revoke a user’s access...................................................................................282
Delete a user.................................................................................................282
Delete a user role..........................................................................................283
Connecting external directory services to PE.................................................283
Connect to an external directory service.......................................................283
External directory settings............................................................................284
Verify directory server certificates..............................................................287
Enable custom password policies through LDAP.....................................288
Working with user groups from an external directory service......................288
Import a user group from an external directory service..............................288
Assign a user group to a user role.................................................................288
Delete a user group......................................................................................289
Removing a remote user’s access to PE.......................................................289
Token-based authentication..........................................................................289
Generate a token using puppet-access..........................................................290
Generate a token using the API endpoint.......................................................292
Use a token with the PE API endpoints.......................................................292
Change the token’s default lifetime..............................................................292
Setting a token-specific lifetime..................................................................293
Set a token-specific label.............................................................................293
Revoking a token..........................................................................................293
Delete a token file.........................................................................................294
Inspecting your infrastructure.........................................................................................327

Monitoring current infrastructure state...........................................................................327
Node run statuses............................................................................................................327
Filtering nodes on the Overview page...........................................................................329
Filtering nodes in your node list....................................................................................331
Monitor PE services..........................................................................................................331
Exploring your catalog with the node graph.................................................................332
How the node graph can help you..................................................................................332
Investigate a change with the node graph....................................................................332
Viewing and managing all packages in use.................................................................333
Enable package data collection.......................................................................................333
View and manage package inventory.............................................................................334
View package data collection metadata.......................................................................334
Disable package data collection....................................................................................334
Infrastructure reports.....................................................................................................335
Working with the reports table.......................................................................................335
Filtering reports.............................................................................................................335
Working with individual reports....................................................................................337
Analyzing changes across Puppet runs.........................................................................338
What is an event?.............................................................................................................338
Event types....................................................................................................................338
Working with the Events page.........................................................................................339
Viewing and managing Puppet Server metrics.............................................................340
Getting started with Graphite.......................................................................................341
Available Graphite metrics..........................................................................................345
Using the developer dashboard......................................................................................349
Status API......................................................................................................................350
Authenticating to the status API....................................................................................350
Forming requests to the status API................................................................................351
JSON endpoints.............................................................................................................351
Activity service plaintext endpoints...............................................................................353
Metrics endpoints..........................................................................................................354
The metrics API............................................................................................................360
Managing nodes................................................................................................................................. 362
  Adding and removing nodes.................................................................................................................... 362
  Managing certificate signing requests.......................................................................................................... 362
  Remove nodes........................................................................................................................................... 363
Running Puppet on nodes................................................................................................................................ 364
  Running Puppet with the orchestrator........................................................................................................ 364
  Running Puppet with SSH............................................................................................................................ 364
  Running Puppet from the console................................................................................................................ 365
  Activity logging on console Puppet runs...................................................................................................... 365
  Troubleshooting Puppet run failures......................................................................................................... 365
Grouping and classifying nodes.................................................................................................................................. 365
  How node group inheritance works............................................................................................................. 366
  Best practices for classifying node groups.................................................................................................. 366
  Create node groups...................................................................................................................................... 366
  Add nodes to a node group............................................................................................................................. 367
  Declare classes............................................................................................................................................ 369
  Enable data editing in the console................................................................................................................ 370
  Define data used by node groups.................................................................................................................. 370
  View nodes in a node group........................................................................................................................... 372
Making changes to node groups.................................................................................................................................. 373
  Edit or remove node groups............................................................................................................................ 373
  Remove nodes from a node group.................................................................................................................. 373
  Remove classes from a node group................................................................................................................. 373
  Edit or remove parameters............................................................................................................................. 373
  Edit or remove variables.................................................................................................................................. 374
Environment-based testing.................................................................................................................................... 374
  Test and promote a parameter....................................................................................................................... 374
  Test and promote a class............................................................................................................................... 374
  Testing code with canary nodes using alternate environments.................................................................... 374
Preconfigured node groups................................................................................................................................... 375
  All Nodes node group.................................................................................................................................... 375
  Infrastructure node groups.............................................................................................................................. 375
  Environment node groups.............................................................................................................................. 378
Designing system configs: roles and profiles............................................................................................................... 379
  The roles and profiles method....................................................................................................................... 379
  Roles and profiles example............................................................................................................................ 382
  Designing advanced profiles.......................................................................................................................... 384
  Designing convenient roles.............................................................................................................................. 401
Node classifier service API.................................................................................................................................... 404
  Forming node classifier requests.................................................................................................................. 405
  Groups endpoint.......................................................................................................................................... 406
  Groups endpoint examples............................................................................................................................. 422
  Classes endpoint.......................................................................................................................................... 424
  Classification endpoint................................................................................................................................. 426
  Commands endpoint.................................................................................................................................... 436
  Environments endpoint................................................................................................................................... 437
  Nodes check-in history endpoint.................................................................................................................... 438
  Group children endpoint............................................................................................................................... 440
  Rules endpoint............................................................................................................................................. 444
  Import hierarchy endpoint............................................................................................................................. 445
  Last class update endpoint............................................................................................................................. 446
  Update classes endpoint................................................................................................................................ 446
  Validation endpoints...................................................................................................................................... 447
  Node classifier errors...................................................................................................................................... 450
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewing jobs</td>
<td>525</td>
</tr>
<tr>
<td>Review jobs from the console</td>
<td>525</td>
</tr>
<tr>
<td>Review jobs from the command line</td>
<td>527</td>
</tr>
<tr>
<td>Puppet orchestrator API v1 endpoints</td>
<td>528</td>
</tr>
<tr>
<td>Puppet orchestrator API: forming requests</td>
<td>529</td>
</tr>
<tr>
<td>Puppet orchestrator API: command endpoint</td>
<td>530</td>
</tr>
<tr>
<td>Puppet orchestrator API: events endpoint</td>
<td>538</td>
</tr>
<tr>
<td>Puppet orchestrator API: inventory endpoint</td>
<td>539</td>
</tr>
<tr>
<td>Puppet orchestrator API: jobs endpoint</td>
<td>542</td>
</tr>
<tr>
<td>Puppet orchestrator API: scheduled jobs endpoint</td>
<td>550</td>
</tr>
<tr>
<td>Puppet orchestrator API: plan jobs endpoint</td>
<td>553</td>
</tr>
<tr>
<td>Puppet orchestrator API: tasks endpoint</td>
<td>558</td>
</tr>
<tr>
<td>Puppet orchestrator API: root endpoint</td>
<td>561</td>
</tr>
<tr>
<td>Puppet orchestrator API: error responses</td>
<td>562</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Managing and deploying Puppet code</td>
<td>562</td>
</tr>
<tr>
<td>Managing environments with a control repository</td>
<td>563</td>
</tr>
<tr>
<td>How the control repository works</td>
<td>563</td>
</tr>
<tr>
<td>Create a control repo from the Puppet template</td>
<td>564</td>
</tr>
<tr>
<td>Create an empty control repo</td>
<td>566</td>
</tr>
<tr>
<td>Add an environment</td>
<td>568</td>
</tr>
<tr>
<td>Delete an environment with code management</td>
<td>568</td>
</tr>
<tr>
<td>Managing environment content with a Puppetfile</td>
<td>569</td>
</tr>
<tr>
<td>The Puppetfile</td>
<td>569</td>
</tr>
<tr>
<td>Managing modules with a Puppetfile</td>
<td>569</td>
</tr>
<tr>
<td>Creating a Puppetfile</td>
<td>569</td>
</tr>
<tr>
<td>Create a Puppetfile</td>
<td>570</td>
</tr>
<tr>
<td>Change the Puppetfile module installation directory</td>
<td>570</td>
</tr>
<tr>
<td>Declare Forge modules in the Puppetfile</td>
<td>570</td>
</tr>
<tr>
<td>Declare Git repositories in the Puppetfile</td>
<td>571</td>
</tr>
<tr>
<td>Managing code with Code Manager</td>
<td>574</td>
</tr>
<tr>
<td>How Code Manager works</td>
<td>574</td>
</tr>
<tr>
<td>Configuring Code Manager</td>
<td>575</td>
</tr>
<tr>
<td>Customize Code Manager configuration in Hiera</td>
<td>580</td>
</tr>
<tr>
<td>Triggering Code Manager on the command line</td>
<td>587</td>
</tr>
<tr>
<td>Triggering Code Manager with a webhook</td>
<td>592</td>
</tr>
<tr>
<td>Triggering Code Manager with custom scripts</td>
<td>594</td>
</tr>
<tr>
<td>Troubleshooting Code Manager</td>
<td>595</td>
</tr>
<tr>
<td>Code Manager API</td>
<td>598</td>
</tr>
<tr>
<td>Managing code with r10k</td>
<td>608</td>
</tr>
<tr>
<td>Configuring r10k</td>
<td>608</td>
</tr>
<tr>
<td>Customizing r10k configuration</td>
<td>610</td>
</tr>
<tr>
<td>Deploying environments with r10k</td>
<td>616</td>
</tr>
<tr>
<td>r10k command reference</td>
<td>618</td>
</tr>
<tr>
<td>About file sync</td>
<td>620</td>
</tr>
<tr>
<td>File sync terms</td>
<td>620</td>
</tr>
<tr>
<td>How file sync works</td>
<td>620</td>
</tr>
<tr>
<td>Checking your deployments</td>
<td>622</td>
</tr>
<tr>
<td>Cautions</td>
<td>622</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Provisioning with Razor</td>
<td>623</td>
</tr>
<tr>
<td>How Razor works</td>
<td>624</td>
</tr>
<tr>
<td>Razor system requirements</td>
<td>626</td>
</tr>
<tr>
<td>Pre-requisites for machines provisioned with Razor</td>
<td>626</td>
</tr>
</tbody>
</table>
SSL and certificates .............................................................................................................. 706
Regenerating certificates: monolithic installs........................................................................ 706
  Regenerate certificates in PE: monolithic installs............................................................... 707
Regenerating certificates: split installs................................................................................ 710
  Regenerate certificates in PE: split installs........................................................................ 710
Individual PE component cert regeneration (split installs only)........................................... 712
  Regenerate Puppet master certs (split installs)................................................................. 712
  Regenerate PuppetDB certs (split installs)........................................................................ 714
  Regenerate PE console certs......................................................................................... 715
Regenerate Puppet agent certificates.................................................................................. 717
Regenerate compile master certs....................................................................................... 718
Use the PE CA as an intermediate CA................................................................................. 719
Managing MCollective................................................................. 724
  MCollective.............................................................................. 725
    Setting up MCollective.......................................................... 725
    Actions and plugins.............................................................. 726
    MCollective orchestration internals....................................... 726
  Invoking actions...................................................................... 727
    Logging into MCollective....................................................... 727
    The mco command................................................................. 727
    Getting help on the command line........................................ 728
    Invoking actions.................................................................... 729
    Filtering actions..................................................................... 730
    Batching and limiting actions............................................... 732
  Controlling Puppet................................................................... 732
    Running Puppet on demand................................................... 732
    Running Puppet on many nodes in a controlled series............. 733
    Enabling and disabling Puppet agent..................................... 733
    Starting and stopping the Puppet agent service.................... 734
    Viewing the Puppet agent's status........................................ 734
    Viewing statistics about recent runs..................................... 734
  List of built-in actions............................................................. 735
  MCollective actions and plugins............................................. 735
  Adding actions and plugins to PE........................................... 745
    Getting new plugins............................................................. 745
    Example plugin class............................................................. 748
  Disabling MCollective............................................................. 749
    Disable MCollective on select nodes..................................... 750
    Disable MCollective on all nodes.......................................... 750
  Change the port used by MCollective/ActiveMQ....................... 750
  Moving from MCollective to Puppet orchestrator..................... 750
    Move from MCollective to Puppet orchestrator..................... 751
  Removing MCollective............................................................ 752

Maintenance............................................................................. 753
  Backing up and restoring Puppet Enterprise............................. 753
    Backup and restore split installations and upgrades............... 754
    Backup and restore monolithic installations and same-version infrastructure........................................... 758
  Puppet Enterprise database maintenance................................ 763
    Databases in Puppet Enterprise............................................ 763
    Optimize a database............................................................... 764
    List all database names........................................................ 764

Troubleshooting....................................................................... 764
  Troubleshooting the installer.................................................. 765
    DNS is misconfigured............................................................ 765
    Security settings are misconfigured...................................... 765
The console was installed before the Puppet master ................................................................. 765
Troubleshooting connections between components ................................................................. 765
  Agents can't reach the Puppet master .................................................................................. 765
  Agents don't have signed certificates .................................................................................. 766
  Agents aren't using the master's valid DNS name ............................................................... 766
  Time is out of sync .............................................................................................................. 766
  Node certificates have invalid dates .................................................................................... 766
  A node is re-using a certname ............................................................................................. 766
  Agents can't reach the filebucket server ............................................................................. 767
Troubleshooting the databases ................................................................................................ 767
  PostgreSQL is taking up too much space ............................................................................ 767
  PostgreSQL buffer memory causes installation to fail ......................................................... 767
  The default port for PuppetDB conflicts with another service ........................................... 768
    puppet resource generates Ruby errors after connecting puppet apply to PuppetDB........ 768
Troubleshooting MCollective .................................................................................................. 768
  ActiveMQ generates errors about too many open files or not enough memory ............... 768
  ActiveMQ doesn't perform as expected ............................................................................. 768
Troubleshooting Windows ....................................................................................................... 769
  Installation fails ................................................................................................................. 769
  Upgrade fails ...................................................................................................................... 769
  Errors when applying a manifest or doing a Puppet agent run ......................................... 769
  Error messages .................................................................................................................. 771
  Logging and debugging ...................................................................................................... 772
Welcome to Puppet Enterprise® 2018.1.16

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>Helpful Puppet Enterprise docs links</th>
<th>Other useful places</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Before you upgrade or install</strong></td>
<td><strong>Docs for related Puppet products</strong></td>
</tr>
<tr>
<td>Release notes - What's new, what's removed, what's resolved.</td>
<td>Open source Puppet</td>
</tr>
<tr>
<td>System requirements - For hardware, software, browsers.</td>
<td>Continuous Delivery for Puppet Enterprise</td>
</tr>
<tr>
<td>What gets installed and where?</td>
<td>Bolt</td>
</tr>
<tr>
<td>System configuration - Set up the network, firewall, and ports.</td>
<td>Puppet Development Kit</td>
</tr>
<tr>
<td><strong>Administer and maintain PE</strong></td>
<td><strong>Why and how people are using PE</strong></td>
</tr>
<tr>
<td>Configuring and tuning PE - Improve performance and scale up.</td>
<td>Read recent blog posts about Puppet Enterprise</td>
</tr>
<tr>
<td>User permissions and roles - Manage access with role-based access control.</td>
<td>Find PE product information</td>
</tr>
<tr>
<td>SSL and certificates - Manage and regenerate security certificates.</td>
<td>Download and try Puppet Enterprise on 10 nodes for free</td>
</tr>
<tr>
<td>Backing up and restoring PE - Prepare for and recover from the worst case scenario.</td>
<td><strong>Learn PE and Puppet</strong></td>
</tr>
<tr>
<td><strong>Learn the basics</strong></td>
<td>Learn at your own pace on a VM</td>
</tr>
<tr>
<td>Getting started for Linux, for Windows - Walk through installing and doing basic operations with PE.</td>
<td>Plan your skills-building path with our learning roadmap</td>
</tr>
<tr>
<td>Accessing the console - Find PE's management tools in the console.</td>
<td>Find an online, in-person or self-paced class</td>
</tr>
<tr>
<td>Inspecting your infrastructure - See what PE knows about your infrastructure.</td>
<td>Get certified</td>
</tr>
<tr>
<td><strong>Manage your IT infrastructure</strong></td>
<td><strong>Get support</strong></td>
</tr>
<tr>
<td>Managing nodes - Use classification, groups, and environments to make changes to your infrastructure.</td>
<td>Search the Support portal and knowledge base</td>
</tr>
<tr>
<td>Roles and profiles - Build a reliable, configurable, refactorable system for your infrastructure.</td>
<td>Find out which PE versions are supported, and for how long</td>
</tr>
<tr>
<td>Application orchestration - Create and manage multi-service and multi-node applications.</td>
<td>Upgrade your support plan</td>
</tr>
<tr>
<td>Running jobs and tasks - Make controlled, on-demand changes.</td>
<td><strong>Share and contribute</strong></td>
</tr>
<tr>
<td>Managing and deploying Puppet code - Use Code Manager to stage, commit, and sync code changes to environments and modules</td>
<td>Engage with the Puppet community</td>
</tr>
<tr>
<td>API index on page 30 - A list of links to documentation for PE and Puppet APIs</td>
<td>Puppet Forge - Find modules you can use, and contribute modules you've made to the community</td>
</tr>
<tr>
<td></td>
<td>Open source projects from Puppet on Github</td>
</tr>
</tbody>
</table>

To send us feedback or let us know about a docs error, open a ticket (you'll need a Jira account) or email the docs team.
PE architecture

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

The following diagram shows the architecture of a monolithic PE installation.

Related information

Component versions in recent PE releases on page 22
This is a historical overview of which components are in Puppet Enterprise (PE) versions, dating back to the previous long term supported (LTS) release.

PE and open source version numbers on page 26
In July 2015, Puppet Enterprise (PE) moved to a new versioning system. This system follows an "x.y.z" pattern, where "x" is the year of the release, "y" is the ordered number of the release within the year, and "z" reflects a patch/bugfix release.

The master of masters and compile masters

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

You can install PE in one of two ways: monolithic or split. In a monolithic installation, the master hosts all services on one node. In a split installation, the services related to the master, the console, and PuppetDB (with PostgreSQL) are each hosted on separate nodes.

Regardless of the installation architecture, the MoM always contains a compile master and a Puppet Server. As your installation grows, you can add additional compile masters to distribute the catalog compilation workload.

Each compile master contains the Puppet Server, the catalog compiler, and an instance of file sync. As your infrastructure grows, you can add compile masters to expand processing capabilities.

See the latest hardware recommendations for more information about choosing a PE installation architecture.
Puppet Server

Puppet Server is an application that runs on the Java Virtual Machine (JVM) on the MoM. 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. See the Puppet Server docs for more information about Puppet Server services.

Catalog compiler

To configure a managed node, Puppet agent uses a document called a catalog, which it downloads from the MoM or a compile master. 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 Puppet code synchronized across multiple compile masters. When triggered by a web endpoint, file sync takes changes from the working directory on the MoM and deploys the code to a live code directory. File sync then deploys that code onto all your compile masters, ensuring that all masters in a multi-master configuration are kept in sync, and that your Puppet code is deployed only when it is ready.

Certificate Authority

The Puppet 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 Puppet ssldir to store credentials. You can use the standard puppet cert 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 MoM or on its own node. The CA service on compile masters is configured, by default, to proxy CA requests to the CA. In addition, you can replace Puppet’s CA with an external CA.

Related information

Hardware requirements on page 148

These hardware requirements are based on internal testing at Puppet and are meant only as guidelines to help you determine your hardware needs.

Installing compile masters on page 205

As your Puppet Enterprise infrastructure scales up to 4,000 nodes and beyond, add load-balanced compile masters to your monolithic installation to increase the number of agents you can manage.

About file sync on page 620

File sync helps Code Manager keep your Puppet code synchronized across multiple masters.

The Puppet agent

Managed nodes run the Puppet agent application, usually as a background service. The MoM and any compile masters also run a Puppet agent.

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

Once 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 will make any changes necessary to correct them. (Or, in no-op mode, it will report on what changes would have been needed.)

After applying the catalog, the agent submits a report to its master. 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**

Puppet agent includes MCollective and Facter.

**MCollective**

MCollective is a distributed task-based orchestration system. Nodes with MCollective listen for commands over a message bus and independently take action when they hear an authorized request. This lets you investigate and command your infrastructure in real time without relying on a central inventory.

**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 `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.

**Related information**

MCollective on page 725

With MCollective in Puppet Enterprise, you can invoke many kinds of actions in parallel across any number of nodes. Several useful actions are available by default, and you can easily add and use new actions.

**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 may want all 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, Inc.

You can also classify nodes using the NC API.

**Related information**

- **Monitoring current infrastructure state** on page 327
  When nodes fetch their configurations from the Puppet master, they send back inventory data and a report of their run. This information is summarized on the **Overview** page in the console.

- **Managing access** on page 275
  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.

- **Endpoints** on page 295
  The role-based access control (RBAC) service enables you to manage users, directory groups, and roles.

- **Activity service API** on page 321
  The activity service logs changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles.

- **Token endpoints** on page 307
  A user's access to PE services can be controlled using authentication tokens. Users can generate their own authentication tokens using the **token** endpoint.

- **Setting PE RBAC permissions and token authentication for orchestrator** on page 478
  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.

- **Request an authentication token for deployments** on page 578
  Request an authentication token for the deployment user to enable secure deployment of your code.

- **Authenticating to the node classifier API** on page 405
  You need to authenticate requests to the node classifier API. You can do this using RBAC authentication tokens or with the RBAC certificate whitelist.

- **Forming RBAC API requests** on page 295
  Web session authentication is required to access the RBAC API. You can authenticate requests by using either user authentication tokens or whitelisted certificates.

- **Forming activity service API requests** on page 321
Web session authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or whitelisted certificates.

**User permissions and user roles** on page 276
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.

**Grouping and classifying nodes** on page 365
Configure nodes by assigning classes, parameters, and variables to them. This is called classification.

**Node classifier service API** on page 404
These are the endpoints for the node classifier v1 API.

### 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 masters, all based on code you keep in Git. They sync the code to your masters, 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 will 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 562
Puppet Enterprise (PE) includes built-in tools for managing and deploying your Puppet infrastructure. Code Manager and r10k are code management tools that automatically install modules, create and maintain environments, and deploy new code to your masters, all based on version control of your Puppet code and data.

**Triggering Code Manager on the command line** on page 587
The puppet-code command allows you to trigger Code Manager from the command line to deploy your environments.

**Running jobs with Puppet orchestrator** on page 473
With the Puppet orchestrator, you can run two types of "jobs": on-demand Puppet runs or Puppet tasks.

### Orchestration services
Orchestration services is the underlying toolset that drives Puppet Application Orchestration and the Puppet orchestrator.

Puppet Application Orchestration provides Puppet language extensions and command-line tools to help you configure and manage multi-service and multi-node applications. Specifically, application orchestration is:

- Puppet language elements for describing configuration relationships between components of a distributed application.

For example, in a three-tier stack application infrastructure—a load-balancer, an application/web server, and a database server—these servers have dependencies on one another. You want the application server to know where the database service is and how they connect, so that you can cleanly bring up the application. You then want the load balancer to automatically configure itself to balance demand on a number of application servers. And if you update the configuration of these machines, or roll out a new application release, you want the three tiers to reconfigure in the correct order.

- A service that orchestrates ordered configuration enforcement from the node level to the environment level.

The orchestrator is a command-line tool for planning, executing, and inspecting orchestration jobs. For example, you can use it to review application instances declared in an environment, or to enforce change on the environment level without waiting for nodes to check in in regular 30-min intervals.
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.

**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>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**

[Puppet Enterprise database maintenance](#) on page 763

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>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>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>Orchestrator</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>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>
<tr>
<td>Razor</td>
<td>HTTPS, SSL</td>
<td>Apache Shiro authentication</td>
<td>Apache Shiro users and roles</td>
</tr>
<tr>
<td>MCollective</td>
<td>SSL</td>
<td>SSL certificate verification with Puppet CA</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**Component versions in recent PE releases**

This is a historical overview of which components are in Puppet Enterprise (PE) versions, dating back to the previous long term supported (LTS) release.

**Puppet Enterprise agent and server components**

This table shows the components installed on all agent nodes.

**Note:**  5 is a backwards-compatible evolution of, which is built into 4.9.0 and higher. To provide some backwards-compatible features, it uses the classic 3.x.x codebase version listed in this table.
<table>
<thead>
<tr>
<th>Version</th>
<th>Agent</th>
<th>Ruby</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.1.16</td>
<td>5.5.21</td>
<td>3.11.14</td>
<td>2.12.5</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.5.20</td>
<td>3.11.13</td>
<td>2.12.5</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.5.19</td>
<td>3.11.12</td>
<td>2.12.5</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.5.18</td>
<td>3.11.11</td>
<td>2.12.5</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.5.17</td>
<td>3.11.10</td>
<td>2.12.5</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.5.16</td>
<td>3.11.9</td>
<td>2.12.4</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.5.15</td>
<td>3.11.8</td>
<td>2.12.4</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.5.10</td>
<td>3.11.7</td>
<td>2.12.4</td>
</tr>
<tr>
<td>2018.1.6</td>
<td>5.5.8</td>
<td>3.11.6</td>
<td>2.12.4</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.5.6</td>
<td>3.11.4</td>
<td>2.12.3</td>
</tr>
<tr>
<td>2018.1.3</td>
<td>5.5.4</td>
<td>3.11.3</td>
<td>2.12.2</td>
</tr>
<tr>
<td>2018.1.2</td>
<td>5.5.2</td>
<td>3.11.2</td>
<td>2.12.2</td>
</tr>
<tr>
<td>2018.1.1</td>
<td>5.5.1</td>
<td>3.11.1</td>
<td>2.12.1</td>
</tr>
<tr>
<td>2017.3.10</td>
<td>5.3.8</td>
<td>3.9.6</td>
<td>2.11.5</td>
</tr>
<tr>
<td>2017.3.9</td>
<td>5.3.7</td>
<td>3.9.6</td>
<td>2.11.5</td>
</tr>
<tr>
<td>2017.3.8</td>
<td>5.3.7</td>
<td>3.9.6</td>
<td>2.11.5</td>
</tr>
<tr>
<td>2017.3.6</td>
<td>5.3.6</td>
<td>3.9.6</td>
<td>2.11.5</td>
</tr>
<tr>
<td>2017.3.5</td>
<td>5.3.5</td>
<td>3.9.5</td>
<td>2.11.4</td>
</tr>
<tr>
<td>2017.3.4</td>
<td>5.3.4</td>
<td>3.9.4</td>
<td>2.11.4</td>
</tr>
<tr>
<td>2017.3.2</td>
<td>5.3.3</td>
<td>3.9.3</td>
<td>2.11.4</td>
</tr>
<tr>
<td>2017.3.1</td>
<td>5.3.2</td>
<td>3.9.2</td>
<td>2.11.3</td>
</tr>
<tr>
<td>2017.3.0</td>
<td>5.3.2</td>
<td>3.9.2</td>
<td>2.11.3</td>
</tr>
<tr>
<td>2017.2.5</td>
<td>4.10.9</td>
<td>3.6.8</td>
<td>2.10.6</td>
</tr>
<tr>
<td>2017.2.4</td>
<td>4.10.8</td>
<td>3.6.7</td>
<td>2.10.5</td>
</tr>
<tr>
<td>2017.2.3</td>
<td>4.10.5</td>
<td>3.6.6</td>
<td>2.10.5</td>
</tr>
<tr>
<td>2017.2.2</td>
<td>4.10.4</td>
<td>3.6.5</td>
<td>2.10.5</td>
</tr>
<tr>
<td>2017.2.1</td>
<td>4.10.1</td>
<td>3.6.4</td>
<td>2.10.4</td>
</tr>
<tr>
<td>2017.1.1</td>
<td>4.9.4</td>
<td>3.6.2</td>
<td>3.3.1</td>
</tr>
<tr>
<td>2017.1.1 for Ubuntu on i386</td>
<td>4.7.0</td>
<td>3.4.1</td>
<td>3.2.1</td>
</tr>
<tr>
<td>2017.1.0</td>
<td>4.9.4</td>
<td>3.6.2</td>
<td>3.3.1</td>
</tr>
<tr>
<td>2017.1.0 for Ubuntu on i386</td>
<td>4.7.0</td>
<td>3.4.1</td>
<td>3.2.1</td>
</tr>
</tbody>
</table>
This table shows components installed on server nodes:

<table>
<thead>
<tr>
<th>Version</th>
<th>Agent</th>
<th>Ruby</th>
<th>OpenSSL</th>
<th>Java</th>
<th>Nginx</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
<tr>
<td>Version</td>
<td>Server</td>
<td>Libs</td>
<td>Java</td>
<td>Nginx</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td></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>
</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>
</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>
</tr>
<tr>
<td>2017.3.10</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.9</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.8</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.6</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.5</td>
<td>5.1.6</td>
<td>5.1.4</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.4</td>
<td>5.1.5</td>
<td>5.1.4</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.2</td>
<td>5.1.4</td>
<td>5.1.3</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.1</td>
<td>5.1.3</td>
<td>5.1.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.3.0</td>
<td>5.1.3</td>
<td>5.1.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.2.5</td>
<td>2.8.0</td>
<td>4.4.2</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.2.4</td>
<td>2.8.0</td>
<td>4.4.2</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.2.3</td>
<td>2.7.2</td>
<td>4.4.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.2.2</td>
<td>2.7.2</td>
<td>4.4.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.2.1</td>
<td>2.7.2</td>
<td>4.4.0</td>
<td>2.5.4</td>
<td>1.6.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.1.1</td>
<td>2.7.2</td>
<td>4.3.2</td>
<td>2.5.1</td>
<td>1.5.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2017.1.0</td>
<td>2.7.2</td>
<td>4.3.2</td>
<td>2.5.1</td>
<td>1.5.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.5.2</td>
<td>2.6.0</td>
<td>4.2.5</td>
<td>2.5.0</td>
<td>1.5.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.5.1</td>
<td>2.6.0</td>
<td>4.2.5</td>
<td>2.5.0</td>
<td>1.5.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.15</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.14</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.13</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.11</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.10</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.9</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.8</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.7</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.6</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.5</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.4</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.3</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.2</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.3</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
<tr>
<td>2016.4.0</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.3</td>
<td>1.4.0</td>
<td>3.1.2</td>
</tr>
</tbody>
</table>
Master and agent compatibility

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

<table>
<thead>
<tr>
<th>Master</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE 3.x</td>
<td>3.x</td>
</tr>
<tr>
<td>Puppet 3.x</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>4.x</td>
</tr>
<tr>
<td></td>
<td>5.x</td>
</tr>
</tbody>
</table>

Note:
- Puppet 3.x has reached end of life and is not actively developed or tested. We retain agent 3.x compatibility with later versions of the master 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>2017.3.0+</td>
<td>version 1, revision 1</td>
</tr>
</tbody>
</table>

PE and open source version numbers

In July 2015, Puppet Enterprise (PE) moved to a new versioning system. This system follows an "x.y.z" pattern, where "x" is the year of the release, "y" is the ordered number of the release within the year, and "z" reflects a patch/bugfix release.

We made this change to align with our release cadence for Puppet Enterprise, which is based on time, rather than number of features. Additionally, time-based versioning makes it easy for you to determine how current your version of PE is.

The first release to use this system was Puppet Enterprise 2015.2. PE 3.8 and earlier will keep their original version numbers.

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.

Open source version numbers

All of our open source projects—including Puppet, PuppetDB, Facter, and Hiera—use semantic versioning ("semver"). This means that in an x.y.z version number, the "y" will increase if new features are introduced and the "x" will increase if existing features change or get removed.

Our semver only refers to the code within that project; it's possible that packaging or interactions with other projects might cause new behavior in a "z" upgrade of Puppet.

Note: In Puppet versions prior to 3.0.0 and Facter versions prior to 1.7.0, we didn't use semver.
Getting support

You can get commercial support for versions of PE in mainstream and extended support. You can also get support from our user community.

Puppet Enterprise support life cycle

Some of our releases have long term support (LTS); others have short term support (STS). For each release, there are three phases of product support: Mainstream support, Extended support, and End of Life (EOL).

For full information about types of releases, support phases and dates for each release, frequency of releases, and recommendations for upgrading, see the Puppet Enterprise support lifecycle page.

If the latest release with the most up-to-date features is right for you, Download, try, or upgrade Puppet Enterprise. Alternatively, download an older supported release from Previous releases.

Open source tools and libraries

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

The customer support portal

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

There are two levels of commercial support plans for PE: Standard and Premium. Both allow you to report your support issues to our confidential customer support portal. When you purchase PE, you'll receive an account and log-on for the portal, which includes access to our knowledge base.

Puppet Enterprise 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, compresses it, and prints the location of the zipped tarball when it finishes running.

The script is provided by the pe_support_script module bundled with the installer. Running the command executes a single bash script which can be found at: /opt/puppetlabs/puppet/cache/lib/puppet_x/puppetlabs/support_script/v1/puppet-enterprise-support.sh

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

Use these options when you run the support script to modify the output:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--render-as &lt;FORMAT&gt;</td>
<td>Specifies what rendering format to use.</td>
</tr>
<tr>
<td>--verbose</td>
<td>Logs verbosely.</td>
</tr>
<tr>
<td>--debug</td>
<td>Logs debug information.</td>
</tr>
<tr>
<td>--classifier</td>
<td>Collects classification data.</td>
</tr>
<tr>
<td>--dir &lt;DIRECTORY&gt;</td>
<td>Specifies where to save the support script's resulting tarball.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--ticket &lt;NUMBER&gt;</td>
<td>Specifies a support ticket number for record-keeping purposes.</td>
</tr>
<tr>
<td>--encrypt</td>
<td>Encrypts the support script's resulting tarball with GnuPG encryption.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You must have GPG or GPG2 available in your PATH in order to encrypt the tarball.</td>
</tr>
<tr>
<td>--log-age</td>
<td>Specifies how many days worth of logs the support script collects. Valid values are positive integers or all to collect all logs, up to 1 GB per log.</td>
</tr>
</tbody>
</table>

**Information collected by the support script**

The support script collects a large amount of system information that can help our Support team troubleshoot issues.

- iptables info: Is it loaded? What are the inbound and outbound rules for both IPv4 and IPv6?
- Facter output generated by `puppet facts find`
- SELinux status
- The amount of free disk and memory on the system
- hostname info: `/etc/hosts` and the output of `hostname --fqdn`
- List of established connections from `netstat`
- The output from `ifconfig`
- The umask of the system
- NTP configuration: what servers are available, the offset from them
- OS and kernel info from `uname -a` and `lsb_release -a` (if installed)
- System package manager configuration files
- The state of installed PE packages as reported by the system package manager
- a list of Ruby gems installed by `/opt/puppetlabs/puppet/bin/gem` and `/opt/puppetlabs/puppetserver/bin/gem`
- the current process list
- a listing of Puppet certs
- a listing of all services and their state (such as running/stopped, enabled/disabled)
- current environment variables
- whether the Puppet master is reachable
- the output of `mco ping` and `mco inventory`
- information about the `peadmin` user used to execute MCollective commands:
  - the client configuration file at: `/var/lib/peadmin/.mcollective`
  - the client log file at: `/var/lib/peadmin/.mcollective.d/client.log`
  - the output of `ulimit -a` for the pe-activemq user along with a count of file descriptors used by the pe-activemq process
  - a list of all Puppet modules on the system
  - the output of `puppet module changes` (shows if any modules installed by PE have been modified)
- a listing (no content) of the files in:
  - `/opt/puppetlabs`
  - `/var/log/puppetlabs`
  - `/var/opt/lib`
- reports the size of the PostgreSQL databases and relations
• PE PostgreSQL configuration from:
  • /opt/puppetlabs/server/data/postgresql/9.6/data
  • the output of SELECT * FROM pg_settings;
  • the output of SELECT * FROM pg_stat_activity;
• The configuration used by PE RBAC to connect with a directory service, excluding sensitive information such as passwords
• r10k information:
  • configuration files
  • output of r10k deploy display -p --detail
• The amount of disk space used by Code Manager
• the output from PE status API endpoints
• output from the PuppetDB /summary-stats endpoint, which provides non-identifying database statistics for troubleshooting
• a list of active nodes from PuppetDB
• a list of active nodes from Orchestration Services
• a histogram of agent run start times from PuppetDB which is used to detect thundering herds
• output from the Puppet Server /environments endpoint, which lists available directory environments and module search paths
• the environment.conf and hiera.yaml configuration files from each environment

It also copies the following files:
• system logs
• output from the dmesg command
• the contents of /etc/puppetlabs/ (being careful to avoid sensitive files, such as modules, manifests, ssl certs, and MCollective credentials)
• configuration files for PE services located under /etc/sysconfig or /etc/default and /etc/ puppetlabs.
• the contents of /var/log/puppetlabs/
• upgrade log files from /opt/puppetlabs/server/data/postgresql/
• the contents of /opt/puppetlabs/puppet/cache/state/
• the contents of /opt/puppetlabs/pe_metric_curl_cron_jobs or /opt/puppetlabs/puppet-metrics-collector, if present
• the contents of /etc/resolv.conf
• the contents /etc/nsswitch.conf
• the contents of /etc/hosts

Getting support from the Puppet community

As a Puppet Enterprise customer you are more than 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 user group. Your request to join will be sent to Puppet, Inc. for authorization and you will receive an email when you’ve been added to the user group.
  • Click on “Sign in and apply for membership.”
  • Click on “Enter your email address to access the document.”
  • Enter your email address.
• Join the open source Puppet user group.
• Join the Puppet developers group.
• Report issues with the open source Puppet project.
**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 the System Configuration documents.

<table>
<thead>
<tr>
<th>API</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<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>Razor API</td>
<td>• Provisioning bare-metal machines.</td>
</tr>
</tbody>
</table>

**Open source Puppet Server, Puppet, PuppetDB, and Forge APIs**

<table>
<thead>
<tr>
<th>API</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet Server administrative API endpoints</td>
<td>• Deleting environment caches created by a Puppet master.</td>
</tr>
<tr>
<td></td>
<td>• Deleting the Puppet Server pool of JRuby instances.</td>
</tr>
<tr>
<td>• environment-cache</td>
<td></td>
</tr>
<tr>
<td>• jruby-pool</td>
<td></td>
</tr>
<tr>
<td>Server-specific Puppet API</td>
<td>• Getting the classes and parameter information that is associated with an environment, with cache support.</td>
</tr>
<tr>
<td>• Environment classes</td>
<td>• Getting information about what modules are installed in an environment.</td>
</tr>
<tr>
<td>• Environment modules</td>
<td>• Getting the contents of a specific version of a file in a specific environment.</td>
</tr>
<tr>
<td>• Static file content</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Useful for</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Puppet Server status API</td>
<td>• Checking the state, memory usage, and uptime of the services running on Puppet Server.</td>
</tr>
<tr>
<td>Puppet Server metrics API</td>
<td>• Querying Puppet Server performance and usage metrics.</td>
</tr>
<tr>
<td>• v1 metrics (deprecated)</td>
<td></td>
</tr>
<tr>
<td>• v2 metrics (Jolokia)</td>
<td></td>
</tr>
<tr>
<td>Puppet HTTP API</td>
<td>• Retrieving a catalog for a node</td>
</tr>
<tr>
<td></td>
<td>• Accessing environment information</td>
</tr>
<tr>
<td></td>
<td>For information on how this API is used internally by Puppet, see Agent/Master HTTPS Communications page</td>
</tr>
<tr>
<td>Certificate Authority (CA) API</td>
<td>• Used internally by Puppet to manage agent certificates.</td>
</tr>
<tr>
<td></td>
<td>See Agent/Master HTTPS Communications for details.</td>
</tr>
<tr>
<td></td>
<td>See Puppet's Commands page for information about the Puppet Cert command line tool.</td>
</tr>
<tr>
<td>PuppetDB APIs</td>
<td>• Querying the data that PuppetDB collects from Puppet</td>
</tr>
<tr>
<td></td>
<td>• Importing and exporting PuppetDB archives</td>
</tr>
<tr>
<td></td>
<td>• Changing the PuppetDB model of a population</td>
</tr>
<tr>
<td></td>
<td>• Querying information about the PuppetDB server</td>
</tr>
<tr>
<td></td>
<td>• Querying PuppetDB metrics</td>
</tr>
<tr>
<td>Forge API</td>
<td>• Finding information about modules and users on the Forge</td>
</tr>
<tr>
<td></td>
<td>• Writing scripts and tools that interact with the Forge website</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 reference manual**

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. Once you’ve followed a link there, you can use the navigation sidebar to browse other sections of the manual.

**Note:** The Puppet manual has information about installing the open source release of Puppet. As a PE user, you should 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 should go in modules. We explain how modules work here.
- There are also guides to installing modules and publishing modules on the Forge.
- As a PE user, you should use the code management features 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, you should ignore those pages. Additionally, the Puppet Server config files in PE are managed with a built-in Puppet module; to change most settings, you should 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.
• You can also write your own custom facts. See the custom fact overview and the custom fact walkthrough.

**Hiera**

• **Hiera docs**

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

Start with the overview 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 walks you through the process of building and executing a query.
• The query structure page explains the fundamentals of using the query API.
• The cURL tips page 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, you should ignore those pages.

**Related information**

Managing and deploying Puppet code on page 562

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

---

**Translated PE**

For both Puppet Enterprise and Puppet, certain resources are available in Japanese, and internationalized for the future addition of other languages.

We provide these resources in Japanese:

• Select interface text, including logs, error messages, and some command-line text. To see translated strings, your system locale and browser language must be set to Japanese, and for the text-based installer, you need gettext. Translated product strings are managed with the disable_i18n setting, and by default, are enabled (puppet Enterprise::master::disable_i18n: false) in PE 2018.1 and later.
• Module READMEs and descriptions for Puppet-supported modules for NTP, Server, Stldlib, AWS, Tomcat, MySQL, Tagmail, PostgreSQL, Apache, and Azure. View the Japanese READMEs on the Puppet Forge by setting your browser language preference to Japanese. You can also find them in your modules directory. The default location is ./readmes/README_ja_JP.md.
• The fully internationalized MySQL module. This is the first module to be released with end-to-end internationalization. To see error and warning messages from the MySQL module in Japanese, set your system locale to Japanese.
• The Puppet Learning VM, an interactive, guided tour that teaches you how to use PE. Both the Quest Guide and the Learning VM itself are available in Japanese. To view the Quest Guide in Japanese, set your system locale to Japanese. To use the Learning VM in Japanese, set your system locale to Japanese. The Quest Guide contains
instructions for changing your locale settings on the virtual machine. The Quest Guide and Learning VM are available at learn.puppet.com

Release notes

These release notes contain important information about Puppet Enterprise® 2018.1.

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 uses certain components of open source Puppet. Refer to the component release notes for information about those releases.

<table>
<thead>
<tr>
<th>Open source component</th>
<th>Version used in PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet Agent</td>
<td>latest</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>latest</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>latest</td>
</tr>
<tr>
<td>Bolt (optional)</td>
<td>latest</td>
</tr>
</tbody>
</table>

Security and vulnerability announcements are posted at https://puppet.com/docs/security-vulnerability-announcements.

Documentation for end-of-life and superseded product versions are archived at https://github.com/puppetlabs/docs-archive.

- **New features** on page 35
  These are the features added to PE 2018.1.x.
- **Enhancements** on page 36
  These are the enhancements added to PE 2018.1.x.
- **Deprecations and removals** on page 44
  These are the features and functions deprecated or removed from PE 2018.1.x.
- **Known issues** on page 45
  These are the known issues in PE in this release.
- **Resolved issues** on page 58
  These are the issues resolved in PE 2018.1.x.
- **What's new since PE 2016.4** on page 65
  If your organization adopts long-term supported (LTS) Puppet Enterprise releases, the release of PE 2018.1 means that you'll soon be upgrading from our previous LTS release, PE 2016.4. This page summarizes the major new features, notable enhancements, deprecation and removals, and high-profile bugs fixed since PE 2016.4 that make 2018.1 a big step forward in your automated configuration management experience.
New features

These are the features added to PE 2018.1.x.

Webhook connection available for Bitbucket configuration (2018.1.8)
When setting up a Code Manager webhook in Bitbucket Server 5.4 or later, you can now configure the connection as a "webhook" instead of a "hook" in Bitbucket configuration.

Continuous Delivery for PE console installation (2018.1.8)
You can now install Continuous Delivery for PE directly from the console using a new Integrations page. Installation leverages a Bolt task requiring a limited set of parameters, so you no longer have to install a separate module or dependencies. For details about installing Continuous Delivery for PE, see Install Continuous Delivery for PE from the PE console in the Continuous Delivery for PE documentation.

puppet infrastructure run command (2018.1.8)
A new puppet infrastructure run command leverages built-in Bolt plans to perform certain PE management tasks, such as regenerating certificates and migrating from a split to a monolithic installation. To use the command, you must be able to connect using SSH from your master 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 master. For information about available plans, run puppet infrastructure run --help.

Enable a new HA replica using a failed master (2018.1.8)
After promoting a replica, you can use your old master as a new replica, effectively swapping the roles of your failed master and promoted replica.

Schedule tasks (2018.1.5)
You can schedule a job to run a task at a particular date and time. Set up a scheduled job from the console or with the endpoint scheduled_jobs that has been added to orchestrator API. For more information, see Schedule a task on page 508.
In addition, Scheduled jobs has been added to the list of permission types available for user roles. You add this permission to user roles that need to delete jobs that another user has scheduled.

Delete user roles (2018.1.3)
You can delete a user role through the console (Access control > User roles). Delete user roles that your organization no longer needs and remove the permissions that the role has given users.

API endpoint for tracking multiple jobs (2018.1.2)
The commands endpoint plan_finish has been added to orchestrator API. You use this endpoint to track jobs that are run together as part of a plan. (A plan combines multiple tasks and runs them with a single command. For more information, see the docs for Bolt.)

API endpoint for checking Code Manager code deployment status (2018.1.2)
This release adds a /deploys/status endpoint for Code Manager. Use this endpoint to check the status of the code deployments that Code Manager is processing. For details about the endpoint, see the Code Manager API docs.

© 2020 Puppet, Inc.
Configure expiration for inactive user accounts (2018.1.2)

The parameters account-expiry-days and password-reset-expiration were added to the RBAC service configuration. You use these parameters to specify the duration, in days, before a user's inactive account expires, and how often, in minutes, the application checks for idle user accounts.

Backup and restore functions (2018.1.0)

This release introduces Puppet Enterprise backup and restore functions. Back up your PE infrastructure, including Puppet code, configuration, PuppetDB, and certificates, allowing you to more easily migrate to a new master or recover from system failures.

Role-based access to tasks (2018.1.0)

Tasks have been added to the list of permission types available for user roles. You assign task permissions to limit user access to specific tasks that run on all nodes or a selected node group.

API endpoints for tracking multiple jobs (2018.1.0)

Two commands endpoints have been added to orchestrator API: plan_start and plan_task. You use these endpoints to track jobs run together as part of a plan. (A plan combines multiple tasks and runs them with a single command. For more information, see the docs for Bolt.)

Related information

Backing up and restoring Puppet Enterprise on page 753
Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

User permissions on page 277
The table below lists the available object types and their permissions, as well as an explanation of the permission and how to use it.

Enhancements

These are the enhancements added to PE 2018.1.x.

Puppet ensures platform repositories aren't installed in order to prevent accidental agent upgrade (2018.1.12)

Previously, Bolt users who installed the Puppet 5 or 6 platform repositories could experience unsupported agent upgrades on managed nodes. With this release, Puppet ensures that the release packages for those platforms are not installed on managed nodes by enforcing ensure => 'absent' for the packages.

Windows install script optionally downloads a tarball of plug-ins (2018.1.12)

For Windows agents, the agent install script optionally downloads a tarball of plug-ins from the master 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.

Note: If your master 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 master agent, which might not match the plug-ins required for agents in a different environment.

This feature is controlled by the setting pe_repo::enable_windows_bulk_pluginsync which you can configure in Hiera or in the console. The default setting for bulk plug-in sync is false (disabled).
**puppet infrastructure run commands no longer require an authentication token (2018.1.12)**

`puppet infrastructure run` commands that affect PuppetDB, including `migrate_split_to_mono` and `enable_ha_failover`, no longer require setting up token-based authentication as a prerequisite for running the command. By default, these commands use the master's PuppetDB certificate for authentication.

**puppet infrastructure run commands provide more useful output (2018.1.12)**

`puppet infrastructure run` commands, such as those for regenerating certificates or enabling high availability failover, provide more readable output, making them easier to troubleshoot.

**Calculations for PostgreSQL settings are fine-tuned (2018.1.12)**

The `shared_buffers` setting uses less RAM by default due to improvements in calculating PostgreSQL settings. Previously, PostgreSQL settings were based on the total RAM allocated to the node it was installed on. Settings are now calculated based on total RAM less the default RAM used by PE services. As a result, on an 8GB installation for example, the default `shared_buffers` setting is reduced from ~2GB to ~1GB.

**Platform support**

This version adds support for these platforms.

**Agent**

- Fedora 31

**The puppet infrastructure run command no longer requires the caserver parameter (2018.1.11)**

The `caserver` parameter is no longer required for `puppet infrastructure run` commands that are run from your master.

**Specify alternate DNS names when regenerating certificates (2018.1.9)**

When regenerating agent or master certificates, you can now pass an optional `dns_alt_names` parameter to the Bolt task.

**puppet backup create skips unnecessary database validation (2018.1.8)**

The `puppet backup create` command now validates your PostgreSQL database only if you specify a `scope` of `puppetdb` or `config`. This change prevents errors if, for example, you specify only `scope=certs`, as you might in a split installation.

**Platform support (2018.1.8)**

This version adds support for these platforms.

**Agent**

- Enterprise Linux 8
- macOS 10.14 Mojave

**Network device agent**

- Enterprise Linux 8

**Improved RBAC API log messages (2018.1.8)**

The RBAC service log entries for revoked users attempting to log in now includes the username and UUID.
**Configuration settings preserved throughout upgrade (2018.1.7)**

Configuration data that you've set in the console and Hiera is backed up every 30 minutes at /etc/puppetlabs/enterprise/conf.d/user_data.conf and restored when you upgrade.

**PE uninstaller no longer uninstalls related products, like Bolt (2018.1.7)**

The PE uninstaller now purges only /opt/puppetlabs/puppet and /opt/puppetlabs/server, leaving behind /opt/puppetlabs if there are any other files remaining in this directory. Leaving this directory in place, as necessary, prevents inadvertently removing other Puppet products, such as Bolt.

**Preconfigured environment node groups (2018.1.7)**

For new installations, we now package a more user-friendly set of preconfigured environment node groups, including:

- All environments
  - Development environment
    - Development one-time run exception
  - Production environment

If you're an existing PE user, your environment node groups are not affected by upgrade.

**Simplified upgrade with high availability enabled (2018.1.7)**

For upgrades from 2017.3 and later with high availability enabled, you now just run the upgrade script on your replica and reinitialize the replica. Upgrades from versions earlier than 2017.3 still require forgetting and recreating the replica in order to update to the latest PostgreSQL version.

**Simplified process for forgetting high availability replicas (2018.1.7)**

The command to forget a high availability replica, puppet infrastructure forget, now includes steps to run Puppet on the master and purge the node as an agent.

**Web-based installation no longer affects known_hosts file (2018.1.7)**

The web-based installer now saves unrecognized SSH signatures in its own file, instead of in the user SSH known_hosts file. Previously, when you used the web-based installation method to install on a remote node that wasn't in the known_hosts file, the node's signature was added to that file.

**Console addition of facts-blacklist-type parameter (2018.1.7)**

The facts-blacklist-type parameter is now a configurable parameter in the console. You can set this parameter to either literal or regex, as appropriate for the values in facts-blacklist.

**Console rebranding (2018.1.7)**

The color palette for Puppet Enterprise has been updated to align with company branding and adhere to W3C Web Content Accessibility Guidelines (WCAG) 2.0.

**Platform support (2018.1.7)**

This release adds support for these platforms.

**Agent**

- Fedora 29
  
  **Note:** MCollective isn't supported on Fedora 29 agents.

© 2020 Puppet, Inc.
Platform support (2018.1.5)
This release adds support for these platforms.

Agent
- SUSE Linux Enterprise Server 15
- Windows Server 2019

Uninstall script on *nix nodes (2018.1.5)
By default, the uninstall script is now copied to all *nix nodes, including infrastructure nodes. You no longer have to manually copy the script from your master when you want to uninstall a *nix node.

*nix bulk plugin sync with the install script (2018.1.5)
For *nix agents, the agent install script can now download a tarball of plugins from the master before the agent runs for the first time. Depending on how many modules you have installed, bulk plugin sync can speed agent installation significantly.

Note: If your master runs in a different environment from your agent nodes, you might see some reduced benefit from bulk plugin sync. The plugin tarball is created based on the plugins running on the master agent, which might not match the plugins required for agents in a different environment.

This feature is controlled by the setting `pe_repo::enable_bulk_pluginsync` which you can configure in Hiera or in the console. Bulk plugin-sync is set to `false` (disabled) by default.

pe.conf synched to high availability replicas (2018.1.5)
The `enterprise/conf.d` directory, including the `pe.conf` file, is now automatically synched from masters to replicas in high availability installations. This expands the available methods for specifying configuration parameters in HA configurations. Previously, we recommended using only Hiera to specify configuration parameters. With this addition, you can now use Hiera or `pe.conf`.

Platform support (2018.1.3)
This release adds support for these platforms.

Master
- Ubuntu 18.04

Agent
- Ubuntu 18.04
- Fedora 28
  
  Note: MCollective isn't supported on Fedora 28 agents.

Agent repositories (2018.1.3)
Agent tarballs for this PE version now contain repositories that better match how open source agents are built and shipped. The `puppet_agent` module version 1.6.2 includes repository updates.

Important: If you manage Debian or Ubuntu nodes using the `puppet_agent` module, you must upgrade to version 1.6.2.

Proxy support parameters added to puppet_enterprise::pxp_agent class (2018.1.3)
Parameters have been added to the class `puppet_enterprise::pxp_agent` to configure the proxy that connects to a master and `pcp-broker`. You can configure these parameters with a proxy URI such as `https://localhost:3128`.

© 2020 Puppet, Inc.
• master_proxy: use to connect to to download task implementations.
• broker_proxy: use to connect to the pcp-broker to listen for task and Puppet runs.

Environment option for backup and restore (2018.1.3)

You can now specify the environment to backup or restore using the puppet-backup command. By default, the puppet-backup command uses --pe-environment=production, so you must use this option to back up and restore if your master is assigned to a different environment than the default production environment.

puppet infra configure behavior (2018.1.3)

This release includes the following enhancements to the way puppet infra configure affects pe.conf:

• Tuning parameters added using Hiera or configuration data in the console are now saved to a file called user_data.conf when you either run puppet infra configure or upgrade. The user_data.conf file, which is saved in the same directory as the pe.conf file, remains in sync with your Hiera and configuration data settings. Previously, parameters were written to pe.conf and remained on your local system even if you removed the parameters from Hiera or configuration data.
• Tuning parameters added to the pe_repo class now persist through upgrades. Previously, parameters were momentarily reverted during upgrades, and then changed back to the correct values.
• Commented-out lines in pe.conf now persist through puppet infra configure commands as well as upgrades. Previously, commented-out lines were removed when you upgraded or ran puppet infra configure.

Simplified console admin password reset (2018.1.3)

This version simplifies resetting the console admin password such that you no longer have to explicitly call the puppet-agent ruby command. This changes the reset script from:

```bash
sudo /opt/puppetlabs/puppet/bin/ruby /opt/puppetlabs/server/bin/set_console_admin_password.rb <NEW_PASSWORD>
```

to:

```bash
sudo /opt/puppetlabs/server/bin/set_console_admin_password.rb <NEW_PASSWORD>
```

Tabs added to Job list page (2018.1.3)

The Job list page uses tabs to organize jobs by type: Puppet run, Task, and Plan. These tabs replace the Job type filter.

Tip: A plan combines multiple tasks and runs them with a single Bolt command. For more information, see Using Bolt with orchestrator on page 482.

View permitted tasks from the command line (2018.1.3)

Run the command puppet task show to view a list of your permitted tasks. To view a list of all installed tasks pass the --all flag: puppet task show --all.

Security (2018.1.2)

A Puppet Query Language (PQL) query has been added to the list of node options available for task permissions. When you assign role-based access to tasks, select PQL query to target nodes that meet specific conditions.
**JRuby (2018.1.0)**

- To support Ruby 2.3, this release changes the PE default setting for JRuby 9k to enabled
  
  (puppet_enterprise::master::puppetserver::jruby_9k_enabled: true). This default differs from open source Puppet and from previous versions of PE.

  **Important**: When upgrading to this version of PE, you must update any server-side installed gems or custom extensions to be compatible with Ruby 2.3 and JRuby 9k. For example, if you're using the autosign gem workflow, upgrade the gem to 0.1.3 and make sure you're not using yardoc 0.8.x. See SERVER-2161 for details.

  To improve performance in installations with multiple JRuby instances or a limited max_requests_per_instance, increase the JVM code cache to 1G (6-12 JRuby instances) or 2G (12-24 JRuby instances), for example
  
  puppet_enterprise::master::puppetserver::reserved_code_cache: 2g.

  If you notice issues with JRuby in PE, file a ticket rather than changing the default jruby_9k_enabled parameter to avoid issues when this setting is eventually deprecated.

**Installation and upgrade (2018.1.0)**

- On Enterprise Linux systems, if you have a proxy between the agent and the master, you can now use the install script to specify an http_proxy_host to be used during package installation, for example
  
  `-s agent:http_proxy_host=<PROXY_FQDN>`. Previously, specifying a proxy host using the install script added the setting to puppet.conf without using it for installation.

- Installer timestamps now include the offset from coordinated universal time (UTC) per ISO 8601 instead of the Java %date format previously used.

- Updates to the Puppet Development Kit (PDK) let you test your modules for compatibility with PE before upgrading, and update or convert modules as needed.

**Preconfigured node groups (2018.1.0)**

- A new PE Infrastructure node group, PE Database, lets you set class parameters to control database configuration. Using the PE Database node group to specify parameters adds the new value to pe.conf and ensures that your settings persist through upgrades.

**Configuration parameters (2018.1.0)**

- The new puppet_enterprise::profile::console::proxy::http_redirect::server_name parameter lets you customize the target URL for HTTP redirects.

- The new puppet_enterprise::ssl_cipher_suites parameter sets the SSL cipher suites for core Puppet services. This parameter expects an array of SSL ciphers, for example:
  

  Console SSL ciphers are managed separately through the new puppet_enterprise::profile::console::ssl_ciphers parameter.

  Cipher names are in RFC format.

- The new puppet_enterprise::profile::database::auto_explain_settings parameter lets you enable and configure auto explain for PE-PostgreSQL using a hash of auto-explain settings. For example:
  
  puppet_enterprise::profile::database::auto_explain_settings: auto_explain.log_min_duration: '10s'
  auto_explain.logVerbose: true

  For details about auto explain settings, see the PostgreSQL documentation.
• The new puppet_enterprise::license::manage_license_key parameter, when set to false, lets you manage your PE license key with your own custom Puppet code, rather than manually copying the license key to /etc/puppetlabs/license.key.

• This version of PE changes the translation default to enabled (puppet_enterprise::master::disable_i18n: false), providing translated logs, reports, and some command-line interface text in Japanese. To see translated strings, your system locale and browser language must be set to Japanese, and for the text-based installer, you need gettext.

  Tip: Previous issues with the translation setting have been resolved, so you no longer need to set environment_timeout to unlimited if translation is enabled; however, doing so can speed catalog compilation.

Orchestrator (2018.1.0)

• You can now run Puppet or tasks from a node group in the console using the Run control. If there are no nodes in a group, the control is deactivated. If you have permission only to run Puppet, or only to run tasks, then the control changes to a button for the specific type of job you can run.

Console (2018.1.0)

• On a package's detail page, you can sort package instances by operating system version and environment.

• Task metadata is available in the console. On the Run a task page, when you select a task to run, you can click view task metadata to see task and parameter information.

• Enhanced behavior of sensitive parameters in tasks. If you mark a parameter as sensitive, its value will not be displayed in logs or API responses when the task runs. In addition, parameters marked as sensitive now appear with the value hidden, so you can rerun the job with the parameter.

• Added conflict detection for node groups. When you commit changes to a node group, you are alerted if another user has made changes to the group while you were editing it. To support this enhancement, two keys have been added to the node classifier service API: serial-number and last-edited.

• Tasks that you run from Puppet Bolt as part of a plan appear in the Job list page, and you can filter them by the job type Plan task. You can view the job details for these tasks; however, to rerun them you must use Bolt. (A plan combines multiple tasks and runs them with a single command. For more information, see the docs for Bolt.)

• A sortable Job ID column appears in the run status table on the Overview page. The ID number of the job a node was most recently part of is displayed in this column.

• On the Permissions tab in the User Roles page, the user permissions object type has been changed from Orchestrator to Job orchestrator and its permissions from View orchestrator to Start, stop and view jobs.

• Puppet run metrics for each node are grouped and available under a Metrics tab. On the Reports page, click a node's Report time, and then click the Metrics tab.

• Enhanced behavior of Event Inspector to work efficiently with large scale deployments. Deduplication of events was removed, and redesigned queries, double filtering, and pagination were added.

Security (2018.1.0)

• You can now revoke and reinstate access to PE for the Administrator account.

• Access controls are hidden for users without permission to use them. Previously, unauthorized users could see these controls but not use them.

• Hostname and wildcard configuration options added to the RBAC directory services to validate that the certificate and the hostname for the connecting client match. To use, set ssl_hostname_validation (defaults to true), and ssl_wildcard_validation (defaults to false). For more information, see Connecting external directory services to PE on page 283.

Razor (2018.1.0)

• The latest Razor client, version 1.8.1, removes the incompatibility with PE and standardizes the client (razor-client) for use with either PE or the open source Razor server.

• Razor now includes built-in tasks for Ubuntu 16.04 and VMware ESXi 6.
• The shiro.ini file used to enable authentication security now uses SHA-256 credential matching by default to specify password hashes.
• A new `has_macaddress_like` tag operator can be used as a regular expression to match hardware MAC addresses.
• New task template helpers `repo_file_contents(PATH)` and `repo_file?(PATH)` can be used to read and check the existence of repo files hosted on mirrors and created with the `--url` argument.
• The `node` task template helper can now be used to evaluate whether a node booted via UEFI, for example `node.hw_hash['fact_boot_type'] == "efi"`.
• This version of Razor documentation contains several improvements, including new details about ensuring scalability in your deployment, and clarifications to the API overview.

Agent support (2018.1.0)

• The agent release included in this version of PE adds support for FIPS 140-2 compliant Enterprise Linux 7

Internationalization (2018.1.0)

• This version of PE changes the translation default to enabled
  `(puppet Enterprise::master::disable_i18n: false)`, providing translated logs, reports, and some command-line interface text in Japanese. To see translated strings, your system locale and browser language must be set to Japanese, and for the text-based installer, you need `gettext`.
  Tip: Previous issues with the translation setting have been resolved, so you no longer need to set `environment_timeout` to `unlimited` if translation is enabled; however, doing so can speed catalog compilation.

Support script (2018.1.0)

• A new `--encrypt` flag can be used when running the support script to encrypt the resulting tarball with GnuPG encryption.
• A new `--log-age` flag can be used when running the support script to specify how many days worth of logs to collect.

Documentation (2018.1.0)

• We’ve reorganized our Windows docs to align them with the rest of the general PE information. For the most part, interacting with Puppet is the same regardless of your operating system. The goals you have and tasks you do are the same. Where code and commands are different for Windows and *nix, the documentation shows examples for both.
• Work through the refreshed Getting started on Windows section to get up and running with PE. If you are unfamiliar with *nix-style system administration, see Basic tasks and concepts in Windows. For additional Windows-specific information, see Troubleshooting Windows, Managing Windows configurations, and Installing and managing Windows modules.

Related information

Test modules before upgrade on page 224
To ensure that your modules will work with the newest version of PE, update and test them with Puppet Development Kit (PDK) before upgrading.

Infrastructure node groups on page 375
Infrastructure node groups are used to manage PE.

Customize the HTTPS redirect target URL on page 243
By default, the redirect target URL is the same as the FQDN of your master, but you can customize this redirect URL.

Methods for configuring Puppet Enterprise on page 235
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 own certificate to the whitelist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

### Deprecations and removals

These are the features and functions deprecated or removed from PE 2018.1.x.

**Application orchestration (2018.1.16)**

Application orchestration and these corresponding features of the Puppet language are 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**

**Split and large environment installations (2018.1.8)**

The split and large environment installations, where the master, console, and PuppetDB were installed on separate nodes, are no longer recommended. Because compile masters do most of the intensive computing, installing the console and PuppetDB on separate nodes doesn’t substantially improve load capability, and adds unnecessary complexity.

For new installations, we now recommend only monolithic configurations, where the infrastructure components are installed on the master. You can add one or more compile masters and a load balancer to this configuration to expand capacity up to 20,000 nodes, and for even larger installations, you can install standalone PE-PostgreSQL on a separate node. For details about current installation configurations, see Choosing an architecture on page 145. For instructions on migrating from a split installation to a monolithic installation, see Migrate from a split to a monolithic installation on page 228.

**Platforms reaching end of support (2018.1.8)**

These platforms have reached end-of-life and are no longer supported.

**Agent**

- Enterprise Linux 6 and 7 IBM z System
- SUSE Linux Enterprise Server 11 and 12 IBM z System

**Network device agent**

- Cumulus Linux

**Developer dashboard (2018.1.7)**

The developer dashboard was removed in Puppet Server version 5.3.7.

**Platforms reaching end of support (2018.1.7)**

These platforms have reached end-of-life and are no longer supported.

**Agent**

© 2020 Puppet, Inc.
• Fedora 27

**External PostgreSQL option in web-based installation (2018.1.5)**

The option to use an external PostgreSQL instance has been removed from the web-based installer. To install with external PostgreSQL, you must use the text-based installation method.

**Platforms reaching end of support (2018.1.5)**

These platforms have reached end-of-life and are no longer supported.

**Agent**

• Fedora 26

**2018.1 support for MCollective**

2018.1 is the last release to support Marionette Collective, also known as MCollective. While 2018.1 remains supported, will continue to address security issues for MCollective. Feature development has been discontinued. Future releases of will not include MCollective. For more information, see the [Puppet Enterprise support lifecycle](#). To prepare for these changes, migrate your MCollective work to orchestrator to automate tasks and create consistent, repeatable administrative processes. Use orchestrator to automate your workflows and take advantage of its integration with console and commands, APIs, role-based access control, and event tracking.

**Select database user settings (2018.1.0)**

These parameters are deprecated and will be removed in a future version of PE:

• puppet_enterprise::activity_database_user
• puppet_enterprise::classifier_database_user
• puppet_enterprise::orchestrator_database_user
• puppet_enterprise::rbac_database_user

**Platforms reaching end of support (2018.1.0)**

These platforms have reached end-of-life and are no longer supported.

**Master**

• SUSE Linux Enterprise Server 11
• Ubuntu 14.04

**Agent**

• Debian 7
• Fedora 25
• macOS 10.10 and 10.11
• Scientific Linux 5
• Windows Vista

**Known issues**

These are the known issues in PE in this release.
• **Installation and upgrade known issues** on page 46  
These are the known issues for installation and upgrade in this release.

• **High availability known issues** on page 48  
These are the known issues for high availability in this release.

• **PuppetDB and PostgreSQL known issues** on page 49  
These are the known issues for PuppetDB and PostgreSQL in this release.

• **Puppet Server known issues** on page 50  
These are the known issues for Puppet Server in this release.

• **Puppet and Puppet services known issues** on page 51  
These are the known issues for Puppet and Puppet services in this release.

• **Supported platforms known issues** on page 52  
These are the known issues with supported platforms in this release.

• **Configuration and maintenance known issues** on page 53  
These are the known issues for configuration and maintenance in this release.

• **Console and console services known issues** on page 54  
These are the known issues for the console and console services in this release.

• **Orchestration services known issues** on page 55  
These are the known issues for the orchestration services in this release.

• **Permissions known issues** on page 55  
These are the known issues for user permissions and user roles in this release.

• **Code management known issues** on page 55  
These are the known issues for Code Manager, r10k, and file sync in this release.

• **Razor known issues** on page 57  
These are the known issues for Razor in this release.

• **Internationalization known issues** on page 57  
These are the known issues for internationalization and UTF-8 support in this release.

---

**Installation and upgrade known issues**

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

**Java 1.8.181 can cause directory services to fail in upgrade**

Because of changes to LDAP support in Java 8 Update 181, Puppet Enterprise might be unable to connect to external directory services (DS) when you upgrade to PE 2018.1.3 or later. This occurs when the hostname PE uses to communicate with the directory service does not match the CN or one DNS altname of the certificate presented by the directory service.

As a workaround, disable the stricter endpoint verification that is the default in Java.

1. In the console, click **Classification**.
2. In the **PE Infrastructure** node group, select **PE Console**.
3. Click **Configuration** and scroll down to the class `puppet_enterprise::profile::console`.
4. Click the **Parameter** name list and select `java_args`.
5. Add the variable to disable endpoint identification (and keep any existing heap settings):
   
   ```
   {"Xmx": "256m", "Xms": "256m",
   "Dcom.sun.jndi.ldap.object.disableEndpointIdentification": "true" }
   ```
6. Click **Add Parameter** and then commit changes.
7. Run Puppet on the appropriate nodes to apply the change.

---

**Console output is shown when installing in quiet mode**

When running the installer in text mode, adding the `-q` option does not successfully activate quiet mode, and the installation process is logged in the console.
Web-based installation requires a second Puppet run to fully install PE services

Web-based installation includes a single, initial Puppet run, but a second Puppet run is required in order to populate `services.conf`. Until a second Puppet run completes, the Puppet service status in the console reports that one or more services isn't accepting requests.

Ubuntu 16.10 agents can't be installed with package management

Installation using package management isn't currently supported. If you need to install an Ubuntu 16.10 agent, log into the node where you want to install the agent and run:

```
wget https://apt.puppetlabs.com/puppetlabs-release-pc1-yakkety.deb
sudo dpkg -i puppetlabs-release-pc1-yakkety.deb
```

Installer can fail due to SSL errors with AWS

In some cases, some master platforms have received SSL errors when attempting to connect to s3.amazonaws.com, and so have been unable retrieve the agent packages needed for installation. Updating the CA cert bundle on the master platform usually resolves the problem. To update the bundle, run the following commands:

```
rm /etc/ssl/certs/ca-bundle.crt
yum reinstall ca-certificates
```

After you update the CA bundle, run the installer again.

Incorrect credentials for console databases cause split upgrade to fail

During a split upgrade, if you supply incorrect credentials for the database associated with the console, including database names, user names, or passwords, the upgrade process fails with an error message. Verify that you're using the correct database credentials and re-run the upgrader. The credentials can be found on the PuppetDB node at `/etc/puppetlabs/installer/database_info.install`.

Web-based installer fails to acknowledge failed installs due to low RAM

When installation fails because a system is not provisioned with adequate RAM, the web-based installer stops responding, but the Start using Puppet Enterprise button is available, suggesting that installation succeeded. In these cases, the command line shows an out of memory: Kill process error.

Provision the system with adequate RAM and re-run the installation.

Hard tabs for indentation in Hiera YAML files cause errors after upgrading

Before upgrade, ensure that any Hiera YAML files do not use hard tabs for indentation.

Incorrect umask value can cause installation and upgrade to fail

Set an umask value of 0022 on your master.

Upgrade fails if autosign.conf contains invalid entries

Entries in `/etc/puppetlabs/puppet/autosign.conf` that don't conform to autosign requirements cause the upgrade to fail when configuring the console. Correct any invalid entries before upgrading. For more information about autosigning, see Config files: autosign.conf.

Install agents with different OS when master is behind a proxy

If your master uses a proxy server to access the internet, you might not be able to download `pe_repo` packages for the agent.

If you're using a proxy, follow this workaround:
1. From your master, navigate to `/etc/sysconfig/`, and create a file called `puppet` with this code:

   ```
   export http_proxy <YOUR_PROXY_SERVER>
   export https_proxy <YOUR_PROXY_SERVER>
   ```

2. Save and close the file, and then restart the puppet service:

   ```
   puppet resource service puppet ensure=stopped
   puppet resource service puppet ensure=running
   ```

3. Repeat these steps on any compile masters in your environment.

**Puppet stdlib functions handle undef too leniently**

PE 2018.1 uses Puppet 5 stdlib functions. These functions handle `undef` less strictly than they should. In Puppet 6, many functions from stdlib were moved into Forge modules, and now treat `undef` more strictly. Consequently, in PE 2019, some code that relies on `undef` values being implicitly treated as other types will return an evaluation error. For more information on which functions were moved out of stdlib, see the Puppet 6.0 release notes.

**High availability known issues**

These are the known issues for high availability in this release.

**Provisioning a replica node that's connected to a compile master causes a run to fail**

If you try to provision a replica node that's connected to a compile master, rather than a primary master, it can cause an error:

```
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/
primary_master_replica.ppLine: 64
```

As a workaround, on the replica you want to provision, edit `/etc/puppetlabs/puppet.conf` so that the `server` and `server_list` settings use a primary master, rather than a compile master.

**Executing enable command immediately after provisioning can cause error**

If you scripted or automated the process to set up a replica, the `enable` command can sometimes fail with the error:

```
ERROR [p.p.puppet] Received a PCP error message due to message 7f7ed934-d2ed-4d64-8b5c-e2cb48cb2227 (blocking request for 'status query')
```

To resolve the issue, run `puppet agent -t` on both the master and replica nodes, then try the `enable` command again.

**Running puppet infrastructure configure fails on a promoted replica**

The `/etc/puppetlabs/enterprise` directory contains configuration files used during installation and upgrade. These files are not automatically copied to a replica; this poses no issue during normal operation, but might cause problems if you need to upgrade on the replica after it is promoted.

To avoid potential issues:

1. Back up the `/etc/puppetlabs/enterprise` directory of the primary master.
2. When you promote a replica, copy the backup into place.
After HA enablement, both server and server_list are set

When the agent configuration file contains settings for both server and server_list, a warning appears. This warning can occur after enabling a replica in a high availability configuration. You can ignore the warning, or hide it by removing the server setting from the agent configuration, leaving only server_list.

Running enable starts an orchestrator run that fails

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.

Latency over WAN can cause failure

If the master 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.

PuppetDB and PostgreSQL known issues

These are the known issues for PuppetDB and PostgreSQL in this release.

Incorrect password for database users causes install/upgrade to fail

If you supply an incorrect password for one of the databases users (RBAC, console, PuppetDB), the installation or upgrade fails. However, in some cases it might appear that the installation or upgrade was successful. For example, if the incorrect password is supplied for the console database user, the installation or upgrade continues, and appears to succeed, but the console isn't functional.

Errors related to stopping pe-postgresql service

If for any reason the pe-postgresql service is stopped, agents receive several different error messages, for example:

Warning: Unable to fetch my node definition, but the agent run will continue:
Warning: Error 400 on SERVER: (<unknown>): mapping values are not allowed in this context at line 7 column 28

or, when attempting to request a catalog:

Error: Could not retrieve catalog from remote server: Error 400 on SERVER: (<unknown>): mapping values are not allowed in this context at line 7 column 28
Warning: Not using cache on failed catalog
Error: Could not retrieve catalog; skipping run

If you encounter these errors, simply re-start the pe-postgresql service.

Mismatch between classifier classification and matching nodes for regexp rules

PuppetDB’s regular expression matching has surprising behaviors for structured fact value comparisons. For example, the structured fact os is a rule that matches 
"~", "os", ":". PuppetDB would unintentionally match every node that has the os structured fact because the regular expression is applied to the JSON encoded version of the fact value.

The classifier does not use PuppetDB for determining classification and regular expressions in the classifier rules syntax only support direct value comparisons for string types.

This has caused issues in the console where the node list and counts for the "matching nodes" display sometimes indicated that nodes were matching even though the classifier would not consider them matching.
After the fix, the same criteria will be applied to the displays and counts that the classifier uses. The output of the classifier’s rule translation endpoints will also make queries that match the classifier behavior.

**Note:** This fix does not change the way nodes are classified, it only corrects how the GUI displays matching nodes.

**Puppet Server known issues**

These are the known issues for Puppet Server in this release.

**File sync error causes pe-puppetserver service to shut down due to incorrect pe-puppet owner and group permissions**

During an r10k deployment, if the pe-puppet user does not have permissions to delete files in the environment directory, a file sync error causes the pe-puppetserver service to shut down during deployment.

This issue can be fixed by setting permissions in the environment directory to owner pe-puppet and group pe-puppet with the command: `chown -R pe-puppet:pe-puppet /etc/puppetlabs/code/environments`.

**Installing gems when Puppet Server is behind a proxy requires manual download of gems**

If you run Puppet Server behind a proxy, the `puppetserver gem install` command fails. Instead you can install the gems as follows:

1. Use rubygems.org to search for and download the gem you want to install, and then transfer that gem to your master.
2. Run `/opt/puppetlabs/bin/puppetserver gem install --local <PATH TO GEM>`.

**Puppet Server run issue when /tmp/ directory mounted noexec**

In some cases, especially for RHEL 7 installations, if the `/tmp` directory is mounted as noexec, Puppet Server might fail to run correctly, and you might see an error in logs similar like:

```
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
```

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.

If you want to use a directory other than `/tmp`, you can add an extra argument to the `$java_args` parameter of the `puppet_enterprise::profile::master` class using the console, for example, add `{"Djava.io.tmpdir=/var/tmp":""}` as the value for the `$java_args` parameter.

`JAVA_ARGS in /etc/default/pe-puppetserver` should have the same value as before.

Whether you use `/tmp` or a different directory, you must 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.

**HTTPS client issues with newer Apache mod_ssl servers**

When configuring Puppet Server to use a report processor that involves HTTPS requests, you might encounter compatibility issues between the JVM HTTPS client and certain server HTTPS implementations. This is usually indicated by a `Could not generate DH keypair` error in the Puppet Server logs. See SERVER-17 for workarounds.

© 2020 Puppet, Inc.
Puppet and Puppet services known issues

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

Restart shell after install for PE client tools subcommands

After installing PE, the commands in the PE client tools aren't available on the PATH until you restart your shell.

Change to lsbmajdistrelease fact affects some manifests

In Facter 2.2.0, the lsbmajdistrelease fact changed its value from the first two numbers to the full two-number, two-number version on Ubuntu and Amazon Linux systems. This might break manifests that were based on the previous behavior. For example, this fact changed from: 12 to 12.04.

Change allow_no_actionpolicy parameter to enforce MCollective action policies

The MCollective ActionPolicy plugin is installed by default in PE. Within the configuration of MCollective, there is a setting that can be used to enforce the use of this ActionPolicy. By default this setting (plugin.actionpolicy.allow_unconfigured) is hardcoded to 1. This default prevents you from enforcing the use of configured action policies.

To change this setting, use the console to edit the value of the allow_no_actionpolicy parameter of the puppet Enterprise::profile::mcollective::agent class located in the PE MCollective node group. To allow ActionPolicy, enter "0".

Puppet module list --tree shows incorrect dependencies after uninstalling modules

If you uninstall a module with puppet module uninstall <module name> and then run puppet module list --tree, you see a tree that does not accurately reflect module dependencies.

The puppet module command does not support Solaris 10

When attempting to use the puppet module command on Solaris 10, you'll get an error like:

```
Error: Could not connect via HTTPS to https://forgeapi.puppetlabs.com
Unable to verify the SSL certificate
The certificate might not be signed by a valid CA
The CA bundle included with OpenSSL might not be valid or up to date
```

This error occurs because there is no CA-cert bundle on Solaris 10 to trust the Forge certificate. To work around this issue, download directly from the Forge website and then use the puppet module install command to install from a local tarball.

Hiera-eyaml gem does not include an executable

The hiera-eyaml gem was included with Puppet agent 5.5.0, but is not installed to a location in the working environment's PATH until Puppet agent 5.5.7, which can lead to unexpected Puppet agent failures:

```
Error: Could not find command 'eyaml'
Error: /Stage[main]/Hiera::Eyaml/Exec[createkeys]/returns: change from 'notrun' to ['0'] failed: Could not find command 'eyaml'
```

The preferred solution is to update Puppet to 5.5.7 or newer. To work around this issue on Puppet agent 5.5.0 to 5.5.6, link the installed hiera-eyaml gem executable to that path:

```
ln -s /opt/puppetlabs/puppet/lib/ruby/vendor_gems/bin/eyaml /opt/puppetlabs/puppet/bin/eyaml
```
Puppet stdlib functions handle `undef` too leniently

PE 2018.1 uses Puppet 5 stdlib functions. These functions handle `undef` less strictly than they should. In Puppet 6, many functions from stdlib were moved into Forge modules, and now treat `undef` more strictly. Consequently, in PE 2019, some code that relies on `undef` values being implicitly treated as other types will return an evaluation error. For more information on which functions were moved out of stdlib, see the Puppet 6.0 release notes.

Supported platforms known issues

These are the known issues with supported platforms in this release.

Ubuntu 16.10 agents can't be installed with package management

Installation using package management isn't currently supported. If you need to install an Ubuntu 16.10 agent, log into the node where you want to install the agent and run:

```
wget https://apt.puppetlabs.com/puppetlabs-release-pc1-yakkety.deb
sudo dpkg -i puppetlabs-release-pc1-yakkety.deb
```

PXP agent stops after upgrading Solaris agents

After upgrading a Solaris agent, the PXP agent service might not restart properly. To restart the agent, run Puppet again or wait until the next timed run.

Readline version issues on AIX agents

- AIX 5.3 agents depend on readline-4-3.2. You can check the installed version of readline by running `rpm -q readline`. If you need to install it, you can download it from IBM. Install it before installing the agents.
- On AIX 6.1 and 7.1, the default version of readline, 4-3.2, is insufficient. You must replace it before upgrading or installing by running:

```
rpm -e --nodeps readline
rpm -Uvh readline-6.1-1.aix6.1.ppc.rpm
```

If you see an error message after installing readline, you can disregard it.

Solaris updates might break Puppet install

In some cases, agents on Solaris platforms quit responding after the core Solaris OS was updated. Essential configuration files were erased from the `/etc/` directory (which includes SSL certs needed to communicate with the master), and the `/etc/hosts` file was reverted.

If you encounter this issue, log in to the master and clear the agent cert from the Solaris machine (`puppet cert clean <HOSTNAME>`), and then re-install the agent.

Solaris 10 and 11 have no default symlink directory

Solaris 10 and 11 don't have the `symlink` directory in their path by default. If you use one of these two platforms, add `/usr/local/bin` to your default path.

Debian and Ubuntu local hostname issue

On some versions of Debian and Ubuntu, the default `/etc/hosts` file contains an entry for the machine's hostname with a local IP address of 127.0.1.1. This can cause issues for PuppetDB and PostgreSQL, because binding a service to the hostname causes it to resolve to the local-only IP address rather than its public IP. As a result, nodes (including the console) fail to connect to PuppetDB and PostgreSQL.

To fix this, add an entry to `/etc/hosts` that resolves the machine's FQDN to its `public` IP address. Complete this step before installing PE. If PE has already been installed, restart or reload the `pe-puppetdb` and `pe-postgresql` services after adding the entry to the hosts file.

© 2020 Puppet, Inc.
**PE can't locate Samba init script for Ubuntu 14.04**

If you attempt to install and start Samba using resource management, you might encounter these errors:

```
Error: /Service[smb]: Could not evaluate: Could not find init script or upstart conf file for 'smb'
Error: Could not run: Could not find init script or upstart conf file for 'smb'
```

To work around this issue, install and start Samba with these commands:

```
puppet resource service smbd provider=init enable=true ensure=running
puppet resource service nmbd provider=init enable=true ensure=running
```

Or, add the Samba service to a manifest:

```puppet
service { 'smbd':
  ensure   => running,
  enable   => true,
  provider => 'init',
}

service { 'nmbd':
  ensure   => running,
  enable   => true,
  provider => 'init',
}
```

**Augeas file access issue**

On AIX agents, the Augeas lens is unable to access or modify `/etc/services`. There is no known workaround.

**Variables are expanded in Windows systems path**

Puppet automatically expands variables in a system path.

For example, this path:

```
PATH=%SystemRoot%\System32
```

Is expanded to:

```
PATH=C:\Windows\System32
```

This should not cause any problems.

**Configuration and maintenance known issues**

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

**puppet infrastructure recover_configuration misreports success if specified environment doesn't exist**

If you specify an invalid environment when running `puppet infrastructure recover_configuration`, the system erroneously reports that the environment's configuration was saved.

**Restoring the pe-rbac database fails with the puppet-backup restore command**

When restoring the pe-rbac database, the restore process exits with errors about a duplicate operator family, citext_ops.
To work around this issue:

1. Log into your PostgreSQL instance:

   ```
   sudo su - pe-postgres -s /bin/bash -c "/opt/puppetlabs/server/bin/psql pe-rbac"
   ```

2. Run these commands:

   ```
   ALTER EXTENSION citext ADD operator family citext_ops using btree;
   ALTER EXTENSION citext ADD operator family citext_ops using hash;
   ```

3. Exit the PostgreSQL shell and re-run the backup utility.

**puppet-backup fails if gems are missing from the master's agent environment**

The `puppet-backup` create command might fail if any gem installed on the Puppet Server isn't present on the agent environment on the master. If the gem is either absent or of a different version on the master's agent environment, you'll get the error "command puppet infrastructure recover_configuration failed". To fix this, install missing or incorrectly versioned gems on the master'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 `report.txt` whenever you run a `puppet-backup` command.

To see what gems and their versions you have installed on your Puppet Server, run the command `puppetserver gem list`. To see what gems are installed in the agent environment on your master, run `/opt/puppetlabs/puppet/bin/gem list`.

**Console and console services known issues**

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

**Deactivated nodes shown in Packages page summary counts**

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.

**Reports page summary counts might be inaccurate on first usage**

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.

**Updated modes are shown without leading zero in events**

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.

**Performance issues with the unpin-from-all endpoint**

As number of groups increases, response time can increase significantly with the `unpin-from-all` endpoint.

**Agent time skew results in incorrect error message**

If an agent attempts to check in with the master but the agent has significant time skew, an incorrect error message is returned. This message masks the real cause of the problem, which is the time skew. The error message says: 400 Error: "Attempt to assign to a reserved variable name: 'trusted' on node".
Orchestration services known issues

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

Existing pxp-agents can't run jobs with new orchestrator instances

If you point an existing agent at a new orchestrator instance—for example, you move an agent to a new PE installation—the new orchestrator might reuse job IDs that are still in the pxp-agent's spool directory. In such cases, any time you run a new job, the agent reports the status of a prior job rather than running a new job.

To work around this issue, delete the contents of the pxp-agent's spool directory.

- On *nix agents: /opt/puppetlabs/pxp-agent/spool
- On Windows agents: C:\ProgramData\PuppetLabs\pxp-agent\var\spool

Agents in job runs might request incorrect environment

When you use the orchestrator puppet job command with the environment flag, agent runs participating in the job might request a catalog for a different environment than the one specified in the job.

To work around this issue, use the node classifier in the console to add agents to the environment that are participating in orchestration jobs. Alternatively, you can assign the agent's environment in its configuration file.

Permissions known issues

These are the known issues for user permissions and user roles in this release.

Code management known issues

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

Default SSH URL with TFS fails with Rugged error

Using the default SSH URL with Microsoft Team Foundation Server (TFS) with the rugged provider causes an error of "unable to determine current branches for Git source." This is because the rugged provider expects an @ symbol in the URL format.

To work around this error, replace ssh:// in the default URL with git@

For example, change:

ssh://tfs.puppet.com:22/tfs/DefaultCollection/Puppet/_git/control-repo

to

git@tfs.puppet.com:22/tfs/DefaultCollection/Puppet/_git/control-repo

GitHub security updates might cause errors with shellgit

GitHub has disabled TLSv1, TLSv1.1 and some SSH cipher suites, which might cause automation using older crypto libraries to start failing. If you are using Code Manager or r10k with the shellgit provider enabled, you might see negotiation errors on some platforms when fetching modules from the Forge. To resolve these errors, switch your configuration to use the rugged provider, or fix shellgit by updating your OS package.

Timeouts when using --wait with large deployments or geographically dispersed compile masters

Because the --wait flag deploys code to all compile masters before returning results, some deployments with a large node count or compile masters spread across a large geographic area might experience a timeout. Work around this issue by adjusting the timeouts_sync parameter.
r10k with the Rugged provider can develop a bloated cache

If you use the rugged provider for r10k, repository pruning is not supported. As a result, if you use many short-lived branches, over time the local r10k cache can become bloated and take up significant disk space.

If you encounter this issue, run git-gc periodically on any cached repo that is using a large amount of disk space in the cachedir. Alternatively, use the shellgit provider, which automatically garbage collects the repos according to the normal Git CLI rules.

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 master. If the module repository uses a default branch that is not master, an error occurs. To work around this issue, specify the default branch with the ref: key in your Puppetfile.

Code Manager webhook does not delete branches as expected

If you delete a branch from your control repository, the Code Manager webhook deployment does not immediately delete that environment from the master's staging or live code directories.

Code Manager deletes the environment when it deploys changes to any other environment. Alternately, to delete the environment immediately, deploy all environments manually:

    puppet code deploy --all --wait

After an error during the initial run of file sync, Puppet Server won’t start

The first time you run Code Manager and file sync on a master, an error can occur that prevents Puppet Server from starting. To work around this issue:

1. Stop the pe-puppetserver service.
2. Locate the data-dir variable in /etc/puppetlabs/puppetserver/conf.d/file-sync.conf.
3. Remove the directory.
4. Start the pe-puppetserver service.

Repeat these steps on each master exhibiting the same symptoms, including the master of masters and any compile masters.

Puppet Server crashes if file sync can’t write to the live code directory

If the live code directory contains content that file sync didn’t expect to find there (for example, someone has made changes directly to the live code directory), Puppet Server crashes.

The following error appears in puppetserver.log:

    org.eclipse.jgit.api.errors.JGitInternalException: Could not delete file /etc/puppetlabs/code/environments/development
    at org.eclipse.jgit.api.CleanCommand.call(CleanCommand.java:138)
    ~[/puppet-server-release.jar:na]

To recover from this error:

1. Delete the environments in code dir: find /etc/puppetlabs/code -mindepth 1 -delete
2. Start the pe-puppetserver service: puppet resource service pe-puppetserver ensure=running
3. Trigger a Code Manager run by your usual method.
Code Manager webhook authentication for GitLab

Prior to GitLab 8.5.0, GitLab had a character limit on webhooks, so the entire Code Manager authentication token could not be passed into GitLab webhooks. If you are using a GitLab version earlier than 8.5.0, either upgrade to the current GitLab version or turn off Code Manager webhook authentication via Hiera with the authenticate_webhook parameter.

File sync error causes pe-puppetserver service to shut down due to incorrect pe-puppet owner and group permissions

During an r10k deployment, if the pe-puppet user does not have permission to delete files in the environment directory, a file sync error causes the pe-puppetserver service to shut down during deployment.

This issue can be fixed by setting permissions in the environment directory to owner pe-puppet and group pe-puppet with this command: chown -R pe-puppet:pe-puppet /etc/puppetlabs/code/environments.

Razor known issues

These are the known issues for Razor in this release.

Temp files aren't removed in a timely manner

Temp files used by Razor might not be removed as quickly as expected. Files are eventually removed when the object is finalized.

Updates might be required for VMware ESXi 5.5 igb files

You might have to update your VMware ESXi 5.5 ISO with updated igb drivers before you can install ESXi with Razor. See the VMware driver package download page for updated igb drivers.

Razor commands resulted in JSON warning

When you run Razor commands, you might get this warning: MultiJson is using the default adapter (ok_json). We recommend loading a different JSON library to improve performance.

You can disregard the warning. However, if you're using Ruby 1.8.7, you can install a separate JSON library, such as json_pure, to prevent the warning from appearing.

pe-razor doesn't allow java_args configuration

Most PE services enable you to configure java_args in the console, but Razor requires you to hard code them in the init script.

Internationalization known issues

These are the known issues for internationalization and UTF-8 support in this release.

Console password must be ASCII

Supported web browsers forbid non-ASCII characters in a login password field. The pe.conf file won't stop you from specifying a non-ASCII password for the console admin password, but you won't be able to use that password to log into the console in your browser.

Only ASCII characters supported in some names in the Puppet language

When naming the following items in Puppet, you must use only ASCII characters:

- Environments
- Variables
• Classes
• Resource types
• Modules
• Parameters
• Tags

Some operating systems forbid non-ASCII names for several types of resources

Some operating systems allow only ASCII characters in the `title` and `namevar` of certain resource types. For example, the user and group resources on RHEL and CentOS can contain only ASCII characters in `title` and `namevar`.

Non-ASCII file names can cause errors when used in concat fragments

When you use concat fragments, an error occurs if a `source` attribute is specified and the `source` attribute points to a file name that contains non-ASCII characters.

Non-ASCII file names can cause errors in the file resource

In a file resource, an error occurs if the `source` attribute is set to a file name containing non-ASCII characters.

Ruby can corrupt the `path` fact and environment variable on Windows

There is a bug in Ruby that can corrupt the environment variable names and values. This bug causes corruption for only some codepages.

The same bug can cause the `path` fact to be cached in a corrupt state.

Resolved issues

These are the issues resolved in PE 2018.1.x.

**Restarting orchestration-services caused pcp-brokers to become unresponsive (2018.1.15)**

When `pe-orchestration-services` restarts on PE masters, the pcp-brokers connected to master no longer fail when attempting to reconnect to the master after it returns.

**Metrics API could leak sensitive information (2018.1.15)**

Puppet Server and PuppetDB could leak sensitive information such as hostnames through the metrics API. This version disables the `/metrics/v1` endpoints by default and restricts access to the `/metrics/v2` endpoints to localhost. (CVE-2020-7943)

**Default for strict_hostname_checking changed to true (2018.1.13)**

The default setting for `strict_hostname_checking` in PE was changed to `true` to resolve a security issue. PE users who are not upgrading should manually set `strict_hostname_checking` to `true` to ensure secure behavior. You must also specify the fully qualified domain name of the host when referring to nodes; partial hostname matches, for example `node /^foo/` are no longer supported.

**Class parameter selection in Firefox had inconsistent behaviors (2018.1.13)**

In the current versions of Firefox, some odd behavior appeared when editing class parameters. The class parameter selection would impact other selections in the class in unexpected ways. As a result, the ability to edit the parameter was removed. You can still edit the parameter value, but cannot choose a different parameter. This resolves the issue in Firefox, simplifies the interaction, and makes the behavior consistent with other pages.
PE ports were vulnerable to LOGJAM (2018.1.13)

PE ports were vulnerable to LOGJAM because they used common, shared Diffident-Hellman primes. The default Diffident-Hellman ephemeral key size has been increased to 2048 for all JVM PE services, like Puppetserver, PuppetDB, etc.

Mismatch between classifier classification and matching nodes for regexp rules (2018.1.12)

PuppetDB’s regular expression matching has surprising behaviors for structured fact value comparisons. For example, the structured fact ‘os’ is a rule that matches ‘{"~", "os", ":"}’. PuppetDB would unintentionally match every node that has the ‘os’ structured fact because the regular expression is applied to the JSON encoded version of the fact value.

The classifier does not use PuppetDB for determining classification and regular expressions in the classifier rules syntax only support direct value comparisons for string types.

This has caused issues in the console where the node list and counts for the "matching nodes" display sometimes indicated that nodes were matching even though the classifier would not consider them matching.

Now, the same criteria are applied to the displays and counts that the classifier uses. The output of the classifier’s rule translation endpoints will also make queries that match the classifier behavior.

**Note:** This fix does not change the way nodes are classified, it only corrects how the GUI displays matching nodes.

Console was inaccessible on macOS Catalina using default certificates (2018.1.12)

Enhanced security requirements in macOS Catalina prevented accessing the console using the default certificate generated during installation.

**puppet infrastructure run commands could fail if the agent was run with cron (2018.1.12)**

`puppet infrastructure run` commands, such as those used for certain installation, upgrade, and certificate management tasks, could fail if the Puppet agent was run with cron. The failure occurred if the command conflicted with a Puppet run.

Replicas tried to query PuppetDB on the primary master (2018.1.11)

In high availability installations, the replica was incorrectly configured to first send queries to the PuppetDB service on the primary master. The failover list has been corrected so that the replica now queries its own PuppetDB service first.

Puppet run failures after a split installation with multiple PuppetDB instances was migrated to a monolithic installation (2018.1.11)

If your split PE installation architecture included multiple standalone PuppetDB instances, the `puppet infrastructure run migrate_split_to_mono` command could not create a PuppetDB instance in the new monolithic installation, and subsequent Puppet runs failed. The `puppet infrastructure run migrate_split_to_mono` command now exits with an error message if multiple PuppetDB instances are present in your split installation.

Rerunning the installer created the All Environments node group (2018.1.11)

If the installer was run for a second time due an issue such as a failed upgrade or a faulty agent lock, the All Environments node group was mistakenly created for the installation. This issue has been resolved, and the All Environments node group is only created for new installations.

**Console output was shown when installing in quiet mode (2018.1.11)**

When running the installer in text mode, adding the `-q` option did not successfully activate quiet mode, and the installation process was logged in the console.
**puppet infra run plans caused Puppet agent service settings to be ignored (2018.1.11)**

In some cases, plans used in `puppet infrastructure run` commands forced the Puppet agent service to run after the plan was complete, even if you had previously disabled the service. The impacted plans now reset the Puppet agent service to the state it was in before the plan was run.

**Package versions were not reset correctly after failed upgrade (2018.1.11)**

If an agent run was in progress as an upgrade began, and consequently the installer failed because it could not acquire an agent lock, the installer did not roll back the relevant packages to the pre-upgrade PE version.

**Upgrade attempts failed when a Puppet run was in progress (2018.1.11)**

If a Puppet run began while the installer was attempting to upgrade PE, conflicts and failures occurred. The installer now checks for Puppet runs before beginning an upgrade, and stops the upgrade if a run is in progress.

**Certificate backup directories could be overwritten (2018.1.11)**

When a certificate regeneration command was run multiple times, certificate backup directories could be unintentionally overwritten. To solve this issue, certificate backup directories are now uniquely named using a time stamp.

**Setting node group environment required Edit configuration data permission (2018.1.10)**

To allow a user role to set a node group environment, users previously had to add the permission `Edit configuration data` in addition to `Set environment`. The permission `Set environment` alone is now enough to allow a user to change the environment.

**Issues discarding changes to RBAC member groups (2018.1.9)**

In previous versions of PE, if you made changes to member groups for a selected user role and later clicked Discard changes instead of Commit changes, the changes were not cleared properly. This has been fixed.

**Replicas did not receive new or updated packages (2018.1.9)**

When new packages were introduced on the master, whether as the result of a PE upgrade or the introduction of new agents, existing replicas did not receive the new packages.

**When removing replicas, empty parameters were added to the PE Agent group (2018.1.9)**

When you run `puppet infrastructure forget` on a high availability deployment that uses load-balanced compile masters, `server_list` and `pcp_broker_list` parameters are no longer added to the `puppetenterprise::profile::agent` class with their values set to empty arrays.

**Upgrade failures caused the pe-installer package to uninstall (2018.1.9)**

If any failure occurred during your upgrade of PE, the `pe-installer` package was automatically uninstalled.

**Reinitializing a replica after upgrade failed (2018.1.9)**

When upgrading a replica, the reinitialize command hung and failed after five minutes depending on the order of services in your `/etc/puppetlabs/client-tools/services.conf` file.

**Enabling a new replica using a previous master failed with autosign enabled (2018.1.9)**

The `puppet infrastructure run enable_ha_failover` command, which lets you enable a failed master as a new replica, includes a step for signing the node's certificate. With autosign enabled, an unsigned certificate couldn't be found, and the command errored out.
Regenerating master certificates failed if the command conflicted with automatic backups (2018.1.9)

The puppet infrastructure run regenerate_master_certificate command failed if it ran at the same time as automatic configuration backups, triggering an error about pre-existing key files.

Regenerating agent certificates failed with autosign enabled (2018.1.9)

The puppet infrastructure run regenerate_agent_certificate command includes a step for signing the node's certificate. With autosign enabled, an unsigned certificate couldn't be found, and the command errored out.

Backup failed with File changed as we read it error (2018.1.9)

If file sync tried to change files while the backup command was archiving files, the backup command errored and failed.

Restore failed if /tmp folder was too small (2018.1.9)

When restoring a master using puppet backup restore, the /tmp folder was always used as the temporary location for unpacking PostgreSQL dumps. If /tmp didn't have enough space to hold these dumps, the restore failed. Additionally, attempting to use a different temp directory by setting the TMPDIR environment variable did not work correctly.

Restore reset master DNS altnames (2018.1.9)

When restoring a master using puppet backup restore, pe_install::puppet_master_dnsaltnames was reset as an array with only the certname of the restore host.

Usernames appear as Base64 strings (2018.1.9)

Under some circumstances when using an LDAP server, usernames that contained extended characters would incorrectly appear as a Base64 encoded string. For example, a username spelled with a German umlaut, Schröder, would appear in the string format U2NocsO2ZGVyDQo=. After upgrading to PE 2018.1.9, users impacted by this issue must log out and log back in to see their usernames correctly spelled.

Puppet file permissions on Windows were modified with every run (2018.1.8)

Changes to how Puppet handled system permissions caused permissions for Windows file resources to be rewritten with each run.

Puppet now treats owner and group on file resources as in-sync in either of these scenarios:

- owner and group are not set in the resource
- owner and/or group are set to the system user on the running node and the system user is set to full control

You can specifically configure the system user to less than full control by setting the owner and/or group parameters to SYSTEM in the file resource. In this case, Puppet emits a warning, because setting SYSTEM to less than full control might have unintended consequences, but it does not modify the permissions.

White space and non-UTF-8 characters in package names and versions triggered invalid byte sequence error (2018.1.8)

When collecting package names and versions, the Windows Package Inspector replaced white spaces and non-UTF-8 characters with "#", causing Puppet runs to fail or submit data that PuppetDB rejected.
Restoring to a former agent node with a signed certificate in the master's ssl/ca/signed directory failed (2018.1.8)

If you attempted to back up data to a former agent node whose node certificate still existed in the `/etc/puppetlabs/puppet/ssl/ca/signed` directory that was part of the backup tarball, the restore failed when it attempted to regenerate a new certificate for the master.

Upgrade could trigger non-blocking errors (2018.1.8)

Upgrade triggered the error `-lt: unary operator expected` when verification of the PostgreSQL version number encountered a `"."

After upgrade, agent runs triggered a cyclical dependency (2018.1.8)

After upgrade, a Puppet run could trigger a cyclical dependency if any of the folders listed below were managed with a file resource on a PE infrastructure node.

- RHEL — `/etc/yum.repos.d`
- Ubuntu — `/etc/apt/sources.list.d`
- SUSE Linux Enterprise Server — `/etc/zypp/repos.d`

You can now manage YUM and APT repositories on infrastructure nodes without encountering cyclical dependency errors from `puppet_enterprise::repo::config`.

Classifier inconsistently applies rules to escaped numerical variables (2018.1.8)

Previously, the node classifier service API inconsistently applied escaped numerical variables to string parameters in class declarations. Where these variables were entered by themselves (`"\$1"`) they were correctly processed. Where these variables were nested in key-value pairs (`{"key" : "a value \$1"}`) they were incorrectly processed as object values. Escaped numerical variables are now processed consistently and correctly.

Script installation failed on Solaris 11.4 agents (2018.1.7)

Due to a change in operating system identification for Solaris 11.4, agent installation using the install script failed with an error indicating the installation method wasn't supported.

Master and replica weren't pinned to appropriate node groups after promotion (2018.1.7)

When a replica was promoted, the retired master remained pinned to the PE Database node group, and the newly promoted master was *not* pinned to the group. Consequently, the database was not automatically managed after you promoted a replica.

Package updates failed on promoted replicas (2018.1.7)

On promoted HA replicas, the managed package repository configuration contains out-of-date references to the old primary master. This causes package updates, like `yum update`, to fail because the repository is pointing to the old master. In general, the packaging configuration artifacts for PE infrastructure are now named `puppet_enterprise` across all platforms. The temporary artifacts created by the `pe_repo` module for agent installation are named `pe_repo`.

Permissions weren't set correctly on global Hiera file (2018.1.7)

On systems with a umask other than 000, the installation process didn't correctly set permissions on the global Hiera file at `/etc/puppetlabs/puppet/global_hiera.yaml`.

Specifying `gc_interval` as an integer caused agent runs to fail (2018.1.7)

If the `puppet_enterprise::profile::puppetdb::gc_interval` parameter was set as an integer, agent runs failed. You can now specify `gc_interval` as either an integer or a string.
Puppet runs generated an autosign warning (2018.1.5)
During Puppet agent runs, a warning appeared indicating that autosign was deprecated. We un-deprecated autosign and this warning no longer appears. For more information about autosigning, see Autosigning certificate requests.

Fedora 26 and 27 agent upgrade failed with module upgrade (2018.1.5)
Using the puppet-agent module to upgrade Fedora 26 and 27 agents fails.

Orchestrator errors for large task payloads (2018.1.5)
Previously, when running tasks that passed arguments larger than approximately 64KB, Orchestrator errors occurred. This was most likely to occur when using Bolt over the Puppet Communications Protocol to upload a file.

Deploying additional code or environments blocked Puppet Server startup (2018.1.5)
Using Code Manager to deploy additional code and environments increased the amount of time it took to start, restart, or reload the pe-puppetserver service, to the point that these operations could time out. The increase in startup time was caused by a diagnostic that checked every file in every environment and logged any modifications. This diagnostic has been removed.

Running r10k deploy on Bionic triggered a Ruby bug (2018.1.5)
Running either r10k 2.6 or 3.0 with the shellgit provider on Bionic could trigger a Ruby bug when running r10k deploy. This bug was more likely the more environments you deploy.

Slow response times when filtering nodes (2018.1.5)
Previously, when you filtered nodes based on fact values (a feature available from the Overview and Classification pages) in an environment with a large number of nodes, it would result in slow response times and, in some cases, cause timeout errors. Now when you filter nodes, a limited number of concurrent node queries is run, which improves performance.

Enumerated task parameter values not available in drop-down lists (2018.1.5)
When a single task parameter value that used the enumerated data type contained a hyphen (-), none of the set values appeared in drop-down lists. For example, these set values Enum[install, status-only, uninstall] would not appear because of the value status-only. This has been fixed.

Requests to the /status/v1/simple API endpoint were denied (2018.1.4)
Requests to the Puppet Server/status/v1/simple endpoint are no longer denied.

Code Manager shut down Puppet Server if a file path contained Java format specifiers (2018.1.4)
When a file name containing the character % was committed to a control repository or module, File Sync interpreted it as a Java format specifier. When this interpretation failed, Code Manager initiated a shutdown of the Puppet Server.

Web-based installation failed MCollective validation (2018.1.4)
Web-based installation still checked for, and failed, MCollective validation even though MCollective was deprecated and no longer installed on new masters beginning in 2018.1.0.

Dynamic node matching using either start- or end-of-line regex erroneously showed no matches (2018.1.3)
When using a start-of-line anchor (^) or an end-of-line anchor ($) to dynamically match nodes in the console, no matches were found, even if matches existed. For example, if you had a node named jupiter, using a regex match ($) for iter shows 1 match, but using a regex match iter$ showed no matches.
Puppet runs were slow when the `puppetlabs-puppetserver_gem` was installed on the master (2018.1.3)

When the `puppetlabs-puppetserver_gem` was installed on the master, Puppet runs became markedly slower, with a long pause before the catalogue was applied.

**MCollective was installed on new high availability replicas (2018.1.3)**

MCollective was installed on new replicas even though the feature was deprecated and no longer installed on new masters beginning in 2018.1.0.

**Enabling package data collection affected MCollective (2018.1.3)**

When enabling package data collection to view a node's current package inventory, MCollective fact filtering would break. This has been fixed.

**Certificate API requests deadlocked high availability replicas (2018.1.3)**

Submitting a CA API request to a replica caused the replica to run out of free worker threads and deadlock.

**Critical corruptive permissions issue with Windows installer for Puppet agents 1.10.13, 5.3.7, and 5.5.2 (2018.1.2)**

Due to a critical issue with the Windows installer, Puppet agents 1.10.13, 5.3.7, and 5.5.2 could inadvertently change permissions on files across a Windows node's filesystem during installation or upgrade.

Upgrades failed on compile masters set to use cached catalogs (2018.1.2)

If `use_cached_catalogs` was set to `true` on a compile master, upgrade failed.

**puppet generate types failed on the `pe_java_ks` module (2018.1.2)**

Running `puppet generate types` on the `pe_java_ks` module resulted in an error.

**PostgreSQL temporary file logging was excessive (2018.1.2)**

PE-PostgreSQL logged an excess of temporary files created by a new PuppetDB query. With this release, temporary files are logged only if they're larger than the size specified by the PostgreSQL `work_mem` setting.

**Options missing from RBAC external directory page (2018.1.2)**

Security protocols, hostname and wildcard validation, and other configuration options have been added to the External directory page in the console. Previously, you were required to configure these options through the directory service (ds) API endpoints.

**Parameters missing from PXP agent class (2018.1.2)**

The parameters `task-download-connect-timeout` and `task-download-timeout` have been added to the class `puppetenterprise::pxp_agent`. For each parameter, set an integer that specifies how much time should pass before the connection or download times out.

**Invalid URL appeared in analytics web service logging (2018.1.2)**

The invalid URL `http://null:null/analytics/v1` has been removed from the web service logging output message.
Slow response times for list of fact values (2018.1.2)

The drop-down list of facts, available when you write rules to assign nodes to a group, has been limited to 500 entries in order to improve response time. In previous releases, installations with large numbers of custom facts sometimes experienced slow response times to view this list.

Slow response times for node classification (2018.1.2)

An internal cache of the classes has been added to improve the response times from the classifier when creating and updating node groups. In previous releases, installations with large numbers of environments sometimes experienced slow response times.

Unquoted hyphens not permitted in identifiers (2018.1.2)

You can use hyphens without quotation marks when you enter identifiers in PE. For example, the following is valid task metadata: {"parameters": {"action": {"type": "Enum[com-start]"}}}. Previously, such an identifier would cause errors.

Disallowed Code Manager proxy setting for the Forge

This release fixes a regression that disallowed the proxy key for the Code Manager forge_settings parameter.

Tasks that produce output with null bytes remain "in progress" after failing (2018.1)

Tasks that produce output with null bytes result in node failures rather than hanging the entire job run.

pcore-java uses java.util.regex instead of jruby/joni (2018.1)

Previously, the type checking for tasks using data types involving regular expressions could be exposed to subtle differences between the regexp implementations in Ruby and the implementations in JVM. The type system support running on Orchestrator has been updated to use the same regular expression implementation as JRuby.

Note: This change might result in patterns accepting input that they did not before. In particular, /\w+/ matches partial strings; to restore the original behavior use anchors, for example, /\A\w+\z/.

Errors when creating node group UUIDs through the API (2018.1)

Previously, if you created a group with a self-generatedUUID using the PUT /v1/groups/:id end point, the classifier would accept some UUIDs that the console considered invalid. As a result the console would generate the error, "Error retrieving group hierarchy: The group id specified is invalid." The regular expression used to validate these identifiers in the console has been made less restrictive to match the classifier behavior. It is: \[0-9a-f\]{8}-[0-9a-f\]{4}-[0-9a-f\]{4}-[0-9a-f\]{4}-[0-9a-f\]{12}.

What's new since PE 2016.4

If your organization adopts long-term supported (LTS) Puppet Enterprise releases, the release of PE 2018.1 means that you'll soon be upgrading from our previous LTS release, PE 2016.4. This page summarizes the major new features, notable enhancements, deprecation and removals, and high-profile bugs fixes since PE 2016.4 that make 2018.1 a big step forward in your automated configuration management experience.

This list provides you with overview of the features and concepts introduced in PE since 2016.4, so you have information you'll need to plan your organization's upgrade to 2018.1. It's not intended to be exhaustive, or to be specific about which short-term supported release each item was added in. For detailed release-by-release breakdowns of all fixes and known issues with Short Term Supported PE releases, see their individual release notes:

<table>
<thead>
<tr>
<th>Individual release notes</th>
<th>2016.5</th>
<th>2017.1</th>
<th>2017.2</th>
<th>2017.3</th>
<th>2018.1</th>
</tr>
</thead>
</table>

© 2020 Puppet, Inc.
Get familiar with the latest hardware requirements, supported operating systems and browsers, and network configuration details in System requirements on page 148.

For information about long and short term support, support levels, and end-of-life dates, see Puppet Enterprise support lifecycle.

New features since 2016.4

High availability for monolithic installations
Ensures that your system remains operational even if certain infrastructure components become unreachable. With high availability enabled, Puppet runs fail over to a replica if your master or master of masters is unreachable. If your master is permanently disabled, you can promote the replica to serve as the new master. See Configuring high availability on page 264.

Backup and restore
Simple commands that back up your PE infrastructure, including Puppet code, configuration, PuppetDB, and certificates, allowing you to more easily migrate to a new master or recover from system failures. For monolithic installations only, not split. See Backing up and restoring Puppet Enterprise on page 753.

Run Puppet from anywhere
Running Puppet isn’t just for the command line anymore. Many pages in the console have Run Puppet buttons that let you launch runs on demand. See Running Puppet on demand on page 489.

Hiera 5
Environment- and module-level data, allowing any environment or module to include its own hiera.yaml file and its own data sources. A new backend system that simplifies integrating custom data sources. Improved performance and better debugging, with complete explanations where Hiera looked and where it found values. A new optional HOCON data backend.

Hiera 5 was brought into the 2016.4 stream in a fix release, so you might be using some of its features already. If you’re still using the deprecated Hiera 4 features, it’s definitely time to move to Hiera 5 instead. Hiera 3 features are still supported. For details, see Upgrading to Hiera 5.

Orchestrator in the console
With orchestrator integrated into the console, you can set up jobs with ease, and use the console’s reporting and infrastructure monitoring tools to review jobs and dig deeper into node run results.

Orchestrator API endpoints for tracking multiple jobs
Two commands endpoints have been added to orchestrator API: plan_start and plan_task. You use these endpoints to track jobs run together as part of a plan. (A plan combines multiple tasks and runs them with a single command. For more information, see the docs for Bolt.)

Packages inventory and management
View a filterable inventory of all packages installed on your nodes, and learn which nodes are using each package version. See which packages are not managed by Puppet and run tasks to update them. Quickly navigate to Puppet code for packages that are managed. See Viewing and managing all packages in use on page 333.

Internationalization support
Improved UTF-8 support, plus a Japanese console, services, and resources. See Translated PE on page 33.

Ad hoc (arbitrary) tasks
You can run arbitrary, one-off tasks from the console, on the command line, or by the orchestrator API, targeting an individual node, a list of nodes, or a set of nodes derived from a PQL query. Watch task execution in real time from the console or command line, and take advantage of built-in job reporting and activity service that shows a complete history of task jobs. Role-based access control allows you to define who can run which tasks on which nodes. See Running tasks on page 503.
Hiera overrides in the console

You can set parameters on node groups without declaring the class. Data that is set in the console is used for automatic parameter lookup, which promotes code regularity and predictability. See Define data used by node groups on page 370.

A new home for the docs

PE docs moved to puppet.com/docs and got somewhat restructured and reorganized. Update your bookmarks! Most old links should automatically redirect to the new locations, but let us know if you can’t find something.

Installation and upgrade enhancements

- On Enterprise Linux systems, if you have a proxy between the agent and the master, you can now use the install script to specify an http_proxy_host to be used during package installation, for example -s agent:http_proxy_host=<PROXY_FQDN>. Previously, specifying a proxy host using the install script added the setting to puppet.conf without using it for installation.
- You can now control the state of the Puppet service when you install *nix or Windows agents with an install script. This capability enables manually kicking off the initial Puppet run or doing so with a provisioning system. Use these flags to control the Puppet service:

<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, stopped</td>
</tr>
<tr>
<td>enable</td>
<td>--puppet-service-enable &lt;VALUE&gt;</td>
<td>-PuppetServiceEnable &lt;VALUE&gt;</td>
<td>true, false, manual, mask</td>
</tr>
</tbody>
</table>

- The simplified agent install script for Windows now supports setting certain MSI properties as flags in the PowerShell script. You can combine agent configurations with MSI properties.

<table>
<thead>
<tr>
<th>MSI Property</th>
<th>PowerShell flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLDIR</td>
<td>-InstallDir</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_USER</td>
<td>-PuppetAgentAccountUser</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_PASSWORD</td>
<td>-PuppetAgentAccountPassword</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_DOMAIN</td>
<td>-PuppetAgentAccountDomain</td>
</tr>
</tbody>
</table>

- PE 2017.3 upgraded PostgreSQL to version 9.6. If you use an external PostgreSQL instance, you must upgrade it before you upgrade PE. If you’re upgrading with high availability enabled, you must upgrade and then forget the existing replica, and provision and enable a new replica.

Ensure you have the right amount of free disk space before you upgrade. Plan for a downtime window of a couple hours if you have a large database, and don’t worry if your upgrade process seems to hang while upgrading the database—it’s not hung. After you upgrade, after everything checks out as working, clean up the old version 9.4 database to free up disk space.

- To support Ruby 2.3, PE 2018.1 changes the default setting for JRuby 9k to enabled (puppet_enterprise::master::puppetserver:::jruby_9k_enabled: true). This default differs from open source Puppet and from previous versions of PE.

**Important:** When upgrading to this version of PE, you must update any server-side installed gems or custom extensions to be compatible with Ruby 2.3 and JRuby 9k. For example, if you’re using the autosign gem workflow, upgrade the gem to 0.1.3 and make sure you’re not using yardoc 0.8.x. For more information about
autosigning, see Autosigning certificate requests. See SERVER-2161 for details on the switch from JRuby 1.7 to JRuby 9k.

If you notice issues with JRuby in PE, file a ticket rather than changing the default parameter setting to avoid issues when this setting is eventually deprecated.

- You can now securely install Windows agents by installing using a certificate.
- On Red Hat, Ubuntu, SUSE Linux Enterprise Server, Solaris, and AIX platforms, if you manually transfer CA certificates to agents and install using the --cacert flag to point to the master CA, subsequent downloads invoked by the installation script are now secured.
- Previously, compile masters downloaded agent packages from puppet.com to make them available for agent installs, meaning they had to reach the internet to retrieve those packages. Compile masters now retrieve agent packages directly from the master of masters.
- Installer timestamps now include the offset from coordinated universal time (UTC) per ISO 8601 instead of the Java %date format previously used.
- Use Puppet Development Kit (PDK) to test your modules for compatibility with PE before upgrading, and update or convert modules as needed.

Configuration enhancements

- A new PE Infrastructure node group, PE Database, enables setting class parameters to control database configuration. Using the PE Database node group to specify parameters adds the new value to pe.conf and ensures that your settings persist through upgrades.
- You can customize the target URL for HTTP redirects using the setting puppet_enterprise::profile::console::proxy::http_redirect::server_name.
- Most of PE’s services (including pe-puppetserver, pe-puppetdb, pe-console-services, and pe-orchestration-services) now have a reload action, which acts like a restart but is significantly faster. If you need to refresh a service after changing its configuration, you can almost always reload it instead of restarting.

To reload a service, run service <NAME> reload instead of service <NAME> restart. The reload action restarts a service without restarting its underlying Java Virtual Machine (JVM) process. Since starting the JVM is the most time-consuming part of a restart, the speed improvement is very noticeable.

Some configuration changes require a full restart. These are:

- Changes to Java arguments, like heap size.
- Changes to Puppet Server’s ca.cfg file, or anything else in a services.d directory.
- A new puppet infrastructure status command displays errors and alerts from PE services, including the activity, classifier, and RBAC services, Puppet Server, and PuppetDB.
- The new puppet_enterprise::ssl_cipher_suites parameter sets the SSL cipher suites for core Puppet services. This parameter expects an array of SSL ciphers, for example:

```
puppet_enterprise::ssl_cipher_suites: ['ECDHE-ECDSA-AES256-GCM-SHA384',
'ECDHE-RSA-AES256-GCM-SHA384', 'ECDHE-ECDSA-CHACHA20-POLY1305']
```

Console SSL ciphers are managed separately through the new puppet_enterprise::profile::console::proxy::ssl_ciphers parameter.

Cipher names are in RFC format.

- The new puppet_enterprise::profile::database::auto_explain_settings parameter lets you enable and configure auto explain for PE-PostgreSQL using a hash of auto-explain settings. For example:

```
puppet_enterprise::profile::database::auto_explain_settings:
  auto_explain.log_min_duration: '10s'
  auto_explain.log_verybose: true
```

For details about auto explain settings, see the PostgreSQL documentation.
• The new puppet_enterprise::license::manage_license_key parameter, when set to false, lets you manage your PE license key with your own custom Puppet code, rather than manually copying the license key to /etc/puppetlabs/license.key.

• This version of PE changes the translation default to enabled (puppet_enterprise::master::disable_i18n: false), providing translated logs, reports, and some command-line interface text in Japanese. To see translated strings, your system locale and browser language must be set to Japanese, and for the text-based installer, you need gettext.

Orchestrator enhancements

• You can run Puppet or tasks from a node group in the console using the Run control. If there are no nodes in a group, the control is deactivated. If you only have permission to run Puppet, or to run tasks, then the control changes to a button for the specific type of job you can run.

• When using the puppet task or puppet job commands on the orchestrator CLI, you can pass PQL queries or node list targets in a text file by specifying the full path to the file prefixed with @ (for example, puppet job run --nodes @/path/to/file.txt). You can also use a text file to pass parameters, in JSON format, when using the puppet task command.

• The puppet job run command now accepts a --description flag. The job description is displayed when you run puppet job show <JOB_ID>.

• In the console, you can now also create jobs from the Overview, Events, and Classification node groups pages, instead of only the Jobs page.

• You can select any job from the job details page and rerun it without having to recreate the settings from the original job. Jobs with application targets run from the command line cannot be rerun from the console, as the console doesn’t support application targets.

• The orchestrator includes a run mode in which you can override an agent's noop = true setting (set in puppet.conf). When you use this run mode, all nodes run in enforcement mode, and a new catalog is enforced on all nodes. In the console, run mode is available as a job setting. On the command line, use the --no-noop flag.

• The Puppet orchestrator communicates with PCP brokers on compile masters on port 8143 and sends job-related messages to the brokers, which are then relayed by the brokers to PXP agents. As you add compile masters, you’re able to scale the number of PCP brokers that can send orchestration messages to agents. See Configure compile masters for orchestration scale for instructions.

• Use the PXP agent log file to debug issues with the Puppet orchestrator. You can change its location from the default as needed.

• The pe-puppetserver service now defaults to an open file limit of 12000 to support orchestrator scale with PCP brokers.

• A new flag, --no-enforce-environment, ensures the orchestrator will ignore the environment set by the --environment flag for agent runs. When you use this flag with the puppet job run command, agents run in the environment specified by the PE Node Manager or their puppet.conf file.

PuppetDB enhancements

• Using the facts_blacklist setting in Hiera, you can now specify a list of facts that should not be stored in the PuppetDB database. See Configure blacklisted facts on page 246.

• Use the puppetdb_query function to query PuppetDB data from Puppet code. See the PQL documentation for more information.

• By default, PuppetDB excludes deactivated or expired nodes from query results. You can change this behavior by setting the node_state query parameter.

• PuppetDB now uses 14 days for a default time-to-live value (node-purge-ttl) before it deletes nodes that have been deactivated or expired. You can change this default behavior.

• PuppetDB now stores incoming commands in stockpile rather than ActiveMQ, increasing performance and reliability.
Console and console services enhancements

- On a package's detail page, you can sort package instances by operating system version and environment.
- Task metadata is available in the console. On the Run a task page, when you select a task to run, you can click view task metadata to open an info pane of task and parameter information.
- Enhanced behavior of sensitive parameters in tasks. If you mark a parameter as sensitive, its value will not be displayed in logs or API responses when the task runs. In addition, parameters marked as sensitive now appear with the value hidden, so you can rerun the job with the parameter.
- Added conflict detection for node groups. When you commit changes to a node group, you are alerted if another user has made changes to the group while you were editing it. To support this enhancement, two keys have been added to the node classifier service API: serial-number and last-edited.
- A sortable Job ID column appears in the run status table on the Overview page. The ID number of the job a node was most recently part of is displayed in this column.
- On the Permissions tab in the User Roles page, the user permissions object type has been changed from Orchestrator to Job orchestrator and its permissions from View orchestrator to Start, stop and view jobs.
- Puppet run metrics for each node are collated and available under a Metrics tab. On the Reports page, click the Report time for a node, and then select the Metrics tab.
- In the console's node detail screen, keys of structured facts are now sorted alphabetically, making them more legible and scannable.
- A new Packages tab on each node's inventory page shows the complete list of installed packages with sortable version, provider, and Puppet management information.
- We've improved console performance by implementing report count caching on the Reports and Overview pages.
- The console now redirects to HTTPS when you attempt to connect over HTTP. The pe-nginx web server now listens on port 80 by default. You can disable the HTTPS redirect in Hiera.
- Previously, the node classifier service stored a check-in for each node when its classification was requested. The check-in included an explanation of how the node matched the rule of every group it was classified into. This functionality created performance issues when managing a large deployment of nodes. The check-in storage is still available, but it's now disabled (false) by default. You can enable this by setting puppet_enterprise::profile::console::classifier_node_check_in_storage to true in the console.
- In this release, you can determine the amount of time that should pass after a node sends its last report before it is considered unresponsive. Set an integer to specify the value in seconds. The default is 3600 seconds (one hour). Adjust puppet_enterprise::console_services::no_longer_reporting_cutoff in the console.
- The ping_interval setting controls how long PXP agents will ping PCP brokers. If the agents don’t receive responses, they will attempt to reconnect. The default is 120 seconds (two minutes). Adjust puppet_enterprise::pxp_agent::ping_interval in the console.
- We’ve redesigned the console’s navigation pane, and reduced its width by half.
- Quickly access the run report associated with a particular event by using the View run report link that now appears on the Events detail page.
- The fact value filters on the Overview and Reports pages now display warning messages if you attempt to use an invalid regular expression, invalid string operator, or empty fact name.
- You are no longer logged out of your console session when you restart the pe-console-services service.
- We have enhanced the usability of the node graph, including updates to controls on the details pane and dependency view.
- Corrective change reporting has been added to the node graph and the Events page in the console.
- The node graph and Events page now provide information regarding whether a Puppet run was completed in enforcement or no-op mode, and whether changes were enforced or simulated.

Code management enhancements

- The new ignore_branch_prefixes subsetting lets you designate specific environments that should not be deployed.
• Two new flags that specify whether local changes to Git modules should be overwritten have been added to r10k. Use `r10k puppetfile install --force` to overwrite local changes when installing updates, and `r10k deploy --no-force` to preserve local changes on deploy.

• A new file sync reset procedure allows you to more easily recover from a failure state.

• We’ve added a new Code Manager parameter, `deploy-ttl`. This parameter specifies the length of time completed deployments are retained before garbage collection, which is important to ensuring consistent Code Manager performance over time.

• We’ve added a new flag, `--dry-run`, to the `puppet-code` command. When you run `puppet-code` with this flag, it tests connections to your control repos and returns a consolidated list of all environments in the control repos.

• The behavior of the `--wait` flag used with the `puppet-code` command has been updated to improve accuracy and completeness of reporting. Previously, `--wait` returned results after deploying code to the code-staging directory. The flag now waits for file sync to also deploy code to the live code directory on all compile masters before returning results.

Due to this updated behavior, running `puppet-code deploy` with `--wait` takes a minimum of 10 seconds longer than in previous versions. In deployments that are geographically dispersed or have a large quantity of environments, completing code deployment can take up to several minutes.

• A `status` action in the `puppet-code` command verifies from the command line that Code Manager and file sync are responding.

• Code Manager provides environment isolation for your resource types. Generated metadata files ensure that each environment uses the correct version of the resource type. For more information, see How Code Manager works on page 574.

RBAC and activity service enhancements

• A user’s activity log now records when they run Puppet from the orchestrator CLI or from the Run Puppet button in the console.

• The `puppet access login` command can now log in using the console `admin` account.

• The Run Puppet on agent nodes permission includes the ability to trigger a Puppet run from the console or orchestrator. See RBAC available permissions.

• The activity service event reporting includes agent runs that are part of orchestration jobs.

Security enhancements

• You can revoke and reinstate access to PE for the Administrator account.

• Administrators can assign permissions to a user role to run tasks on all nodes or a selected node group.

• Removed access controls from the console for users without permission to use them. Previously unauthorized users could view these controls but not use them.

• Hostname and wildcard configuration options added to the RBAC directory services to validate that the certificate and the hostname for the connecting client match.

• Connections to PE databases can now be made only with certificates. User names and passwords are no longer used by default.

• Unlabeled RBAC tokens stored in the database are now hashed. If you label tokens, they are stored unencrypted.

• For those with security compliance needs, PE now supports disabling TLSv1. Services in PE support TLS versions 1, 1.1, and 1.2.

• The MCollective package agent plug-in helps you install packages from any source (including a URL) and does not require that the packages are signed. This provides a `peadmin` user the ability to execute arbitrary code on any MCollective server.

• A default action policy has been put into place in PE that disallows using the package `install`, `uninstall`, and `purge` actions. The policy can be modified and additional action policies can be added using the `puppet_enterprise::profile::mcollective::agent::allowed_actions` parameter to specify agent plug-ins you want to apply an action policy to, and a list of the actions you want to explicitly allow.

• MCollective client keys are labeled sensitive and will not be stored in PuppetDB.
• Use the **certregen module**, available on the Forge, to regenerate and redistribute Puppet CA certificates that are expiring soon, as the Puppet CA cert expires after five years. Refer to the module’s README for full instructions.

**Razor enhancements**

• The latest Razor client, version 1.8.1, removes the incompatibility with PE and standardizes the client (**razor-client**) for use with either PE or the open source Razor server.
• The Razor client is now supported on Windows 2016 Servers.
• Razor now includes supported tasks for SUSE Linux Enterprise Server 11 and 12, Fedora 23, Windows 2016, Ubuntu 16.04, and VMware ESXi 6.
• The shiro.ini file used to enable authentication security now uses SHA-256 credential matching by default to specify password hashes.
• A new **has_macaddress_like** tag operator can be used as a regular expression to match hardware MAC addresses.
• New task template helpers **repo_file_contents (PATH)** and **repo_file? (PATH)** can be used to read and check the existence of repo files hosted on mirrors and created with the `--url` argument.
• The **node** task template helper can now be used to evaluate whether a node booted via UEFI, for example `node.hw_hash['fact_boot_type'] == "efi"`.
• Razor now supports Windows installation using international ISOs. When you create a Windows policy to provision non-English systems, use the **node_metadata** attribute to specify a locale.
• **New commands** let you update an existing policy’s repository, broker, or node metadata. For example, without recreating any policies, you can add a broker to a policy that used the **noop** broker, or switch to the **puppet-pe** broker when you upgrade to Puppet Enterprise.
• In the **create-broker** command, a new **ntpdate_server** property in the **configuration** attribute lets you specify an NTP server. The server is used to synchronize the date and time before installing the agent, which prevents certificate errors.
• In the **modify-node-metadata** command, a new **force** attribute lets you bypass errors in a batch operation with **no_replace**. Existing keys aren’t modified.

**Puppet Server enhancements**

• The **max_requests_per_instance** setting, which controls the maximum number of requests per instance of a JRuby interpreter, increased from 10,000 to 100,000. This change provides a performance boost while still clearing short-lived environments.
• Puppet Server no longer requires a restart or reload in order to enforce certificate revocation.
• Puppet Server now uses Jetty 9.4.
• Puppet Server now uses JRuby 9.1.16.0 or later by default.
• The sample Grafana dashboard for Puppet Server metrics visualizes new metrics.
• Puppet Server now has a **max_queued_requests** setting that can be used to control a **thundering herd problem**.
Analytics enhancements

• Puppet Enterprise collects data about your PE installation and sends it to Puppet so we can improve our product. In addition to previously collected analytics, we now also collect basic information about:
  • agents
  • console usage
  • node groups
  • orchestrator jobs
  • Amazon Web Services Marketplace Image use
  • Puppet Server
  • Cloud platform and hypervisor use
  • All-in-one puppet-agent package version
  • Use of MCollective and non-default user roles
  • JVM memory usage
  • Certificate autosign setting

For details about what data we collect and how to opt out, see Analytics data collection on page 258.

Other enhancements

• In order to keep environment-specific data within the environment folder, the default global hiera.yaml now supports console data only and doesn’t include the previous YAML file hierarchy. If you rely on the default global YAML file hierarchy of nodes/%(clientcert).yaml and common.yaml, create a Hiera 5 compatible hiera.yaml file in your control repo environment folder instead. For example: https://github.com/puppetlabs/control-repo/blob/production/hiera.yaml

• You can now purge nodes without running Puppet on your master and reloading it. However, if you use compile masters, you must still run Puppet on all compile masters in order to revoke a node’s certificate and have the change take effect.

• On non- Windows systems, MCollective server logs now appear in /var/log/puppetlabs/mcollective, consistent with other log files.

• Java garbage collection logs can help you diagnose performance issues with JVM-based PE services. Garbage collection logs are now enabled by default in PE, and the results are captured in the support script, but you can disable them if you need to.

• To help with troubleshooting, you can customize the MCollective client logging level either in the console or in pe.conf by setting puppet_enterprise::profile::mcollective::padmin::mco_loglevel to debug, warning, or error instead of the default info.

Notable bug fixes

• Puppet runs that are halted by provider errors no longer show up as successful in the console.

• RBAC no longer defaults to searching nested LDAP groups. This improves login times when connected to large directory trees.

• The puppet infrastructure configure command no longer hangs if the production environment is missing.

• Numerous issues related to installing PE on nodes behind proxy servers have been fixed.

• We improved PuppetDB performance for structured facts that change often.

Notable deprecations and removals

• MCollective is deprecated. See the important note in the MCollective documentation for details about support and what actions to take if you use MCollective. MCollective is no longer installed by default for new installations. If you use MCollective and you take a migration approach to upgrading—you install the new version from scratch and move new agents over to it—you must enable MCollective either as you install PE 2018.1 or before you migrate the agents.
• Several platforms are no longer supported or are nearing the end of their support by PE. See System requirements on page 148 for the latest supported platforms lists.
• Several parameters involving managing external databases are deprecated:
  - puppet_enterprise::activity_database_user
  - puppet_enterprise::classifier_database_user
  - puppet_enterprise::orchestrator_database_user
  - puppet_enterprise::rbac_database_user
• The following orchestration flags, which were deprecated in previous releases, are removed:
  - puppet-job list is now puppet-job show
  - puppet-job --env is now puppet-job --environment
  - puppet-job App[inst] is now puppet-job -a App[inst]
  - ~/.puppetlabs/etc/puppet/orchestrator.conf is no longer a valid orchestration config file location. Instead use ~/.puppetlabs/client-tools/orchestrator.conf
• The puppet enterprise configure command was renamed to puppet infrastructure configure.
• The whole_environment orchestration target is removed. Jobs with that target cannot be run from the console or command line.
• The unsupported option to disable file sync while keeping Code Manager enabled is removed.
• The file_sync_repo_id and file_sync_auto_commit Code Manager parameters are removed. PE ignores these parameters and raises a warning if you have set them.

Getting started

The content in the following sections is designed to help you get to know Puppet Enterprise.

In these sections, you learn to install PE, manage user permissions, install modules, and finally, write your own module. You also get to try some essential configuration tasks, like time synching with NTP, controlling user permissions with Sudo, or managing firewall rules.

• Getting started on *nix on page 74
Welcome to Puppet Enterprise (PE) getting started for *nix users. Whether you’re setting up a PE installation for actual deployment or want to learn some fundamentals of configuration management with PE, this series of guides provides the steps you need to get up and running relatively quickly.

• Getting started on Windows on page 88
Welcome to Puppet Enterprise (PE) getting started for Windows users. These pages demonstrate some essential tasks to start using PE, either to help you set up an initial installation for actual deployment, or as a way for you to try out some fundamentals of configuration management.

• Common configuration tasks on page 99
This selection of common configuration tasks gives you just a sample of the things you can manage with Puppet Enterprise. These steps provide an excellent introduction to the capabilities of PE.

Getting started on *nix

Welcome to Puppet Enterprise (PE) getting started for *nix users. Whether you’re setting up a PE installation for actual deployment or want to learn some fundamentals of configuration management with PE, this series of guides provides the steps you need to get up and running relatively quickly.

We’ll walk you through the setup of a monolithic (all-in-one node) installation and show you how to automate some basic tasks that sysadmins regularly perform.

The guides present tasks in the order that you would most likely perform them. See the prerequisite sections in each guide to ensure you have the required setup.
• **Start installing PE on *nix** on page 75
This guide uses a web-based installer to walk you through a basic installation. This installation type is ideal for trying out PE with up to 10 nodes, and can be used to manage up to 4000 nodes. This basic setup installs the Puppet master, the PE console, and PuppetDB all on one node, a *nix machine. This guide also reviews prerequisites necessary to a PE installation.

• **Start installing *nix agents** on page 77
When a node is managed by Puppet, it runs a Puppet agent application, commonly called an *agent*. In this guide you'll install a *nix Puppet agent, which regularly pulls configuration catalogs from a Puppet master and applies them locally. This section includes how to sign the agent certificate request in the console.

• **Start installing modules** on page 79
Modules are self-contained bundles of code and data. You can write your own modules, and you can download pre-built modules from the Puppet Forge. In this guide, you'll install a Puppet module, the primary means by which PE configures and manages nodes.

• **Start adding classes** on page 80
In this guide, you'll create a new node group, then apply a named chunk of Puppet code --- better known as a class --- to the node group. You'll apply the apache class to your agent node, which will allow you to launch the default Apache virtual host on your agent node.

• **Start assigning user permissions** on page 81
This section introduces you to role-based access control. You’ll learn how to manage users by creating a user role, granting that role permissions to work with a node group, and assigning users to the role.

• **Start writing modules for *nix** on page 83
In this guide, you’ll modify a Forge module, and you'll use the Puppet Development Kit to write a module and unit test it. This will help you become more familiar with Puppet modules and module development.

• **Next steps** on page 88
After you’ve worked through the Puppet Enterprise (PE) getting started steps, you can now perform the core workflows of a Puppet user.

---

**Start installing PE on *nix**

This guide uses a web-based installer to walk you through a basic installation. This installation type is ideal for trying out PE with up to 10 nodes, and can be used to manage up to 4000 nodes. This basic setup installs the Puppet master, the PE console, and PuppetDB all on one node, a *nix machine. This guide also reviews prerequisites necessary to a PE installation.

On a single *nix machine, you'll install:

- The master, the central hub of activity, where Puppet code is compiled to create agent catalogs, and where SSL certificates are verified and signed.
- The PE console, the web interface, which features numerous configuration and reporting tools.
- PuppetDB, which collects data generated throughout your Puppet infrastructure.

Once this installation is complete, you'll move on to installing Puppet Enterprise on the nodes you wish to manage with Puppet, commonly known as agent nodes.

**Web-based installation prerequisites**

Review these prerequisites and tips before beginning a web-based installation.

- If you’ve previously installed Puppet or Puppet Enterprise, make sure that the machine you're installing on is free of any artifacts left over from the previous installation.

- Make sure that DNS is properly configured on the machines you're installing on.

  - All nodes must know their own hostnames, which you can achieve by properly configuring reverse DNS on your local DNS server, or by setting the hostname explicitly. Setting the hostname usually involves the hostname command and one or more configuration files, but the exact method varies by platform.
  - All nodes must be able to reach each other by name, which you can achieve with a local DNS server, or by editing the /etc/hosts file on each node to point to the proper IP addresses.
• You can run the installer from a machine that is part of your deployment or from a machine that is outside your deployment. If you want to run the installer from a machine that is part of your deployment, in a split installation, run the installer from the node assigned the console component.
• The machine you run the installer from must have the same operating system and architecture as your deployment.
• The web-based installer does not support sudo configurations with `Defaults targetpw` or `Defaults rootpw`. Make sure your `/etc/sudoers` file does not contain, or comment out, those lines.
• For Debian users, if you gave the root account a password during installation of Debian, sudo may not have been installed. In this case, you must either install as root, or install sudo on any nodes on which you want to install.

**Download and verify the installation package**
PE is distributed in downloadable packages specific to supported operating system versions and architectures. Installation packages include the full installation tarball and a GPG signature (.asc) file used to verify authenticity.

**Before you begin**
You must have GnuPG installed.

1. **Download** the tarball appropriate to your operating system and architecture.
2. Import the Puppet public key.
   ```
   wget -O - https://downloads.puppetlabs.com/puppet-gpg-signing-key.pub | gpg --import
   ```
3. Print the fingerprint of the key.
   ```
   gpg --fingerprint 0x7F438280EF8D349F
   ```
   The primary key fingerprint displays: 6F6B 1550 9CF8 E59E 6E46 9F32 7F43 8280 EF8D 349F.
4. **Verify** the release signature of the installation package.
   ```
   $ gpg --verify puppet-enterprise-<version>-<platform>.tar.gz.asc
   ```
   The result is similar to:
   ```
   gpg: Signature made Tue 18 Sep 2016 10:05:25 AM PDT using RSA key ID EF8D349F
   gpg: Good signature from "Puppet, Inc. Release Key (Puppet, Inc. Release Key)"
   ```

   **Note:** If you don't have a trusted path to one of the signatures on the release key, you receive a warning that a valid path to the key couldn't be found.

**Install using web-based configuration**
Web-based installation uses a web server to guide you through installation.

Make sure you have reviewed the prerequisites in a previous section before you begin installing.

1. Unpack the installation tarball:
   ```
   tar -xf <TARBALL_FILENAME>
   ```
2. From the installer directory, run the installer:
   ```
   sudo ./puppet-enterprise-installer
   ```
3. When prompted, choose guided installation

   The installer starts a web server and provides an installer URL.

   The default installer URL is `https://<INSTALL_PLATFORM_HOSTNAME>:3000`. If you forwarded ports in step 1, the URL is `https://localhost:3000`.

4. In a browser, access the installer URL and accept the security request.

   The installer uses a default SSL certificate. You must add a security exception in order to access the installer.

   **Important:** Leave your terminal connection open until the installation is complete or else installation fails.

5. Follow the prompts to configure your installation.

6. On the validation page, verify the configuration and, if there aren't any outstanding issues, click **Deploy now**.

   Installation begins. You can monitor the installation as it runs by toggling **Log View** and **Summary View**. If you notice errors, check `/var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log` on the machine from which you're running the installer.

   When the installation completes, the installer script that was running in the terminal closes.

7. Click **Start using Puppet Enterprise** to log into the console.

**Log into the PE console**

On the validation page, the installer verifies various configuration elements (for example, if SSH credentials are correct, if there is enough disk space, and if the OS is the same for the various components).

1. When you click **Start using Puppet Enterprise**, you receive a browser warning about an untrusted certificate. This is 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 login page for the console, log in with the username `admin` and the password you created when installing the Puppet master.

   Congratulations, you've successfully installed the Puppet master node! Next you'll install PE on your agent nodes, so that you can manage these nodes with Puppet.

**Start installing *nix agents**

When a node is managed by Puppet, it runs a Puppet agent application, commonly called an agent. In this guide you'll install a *nix Puppet agent, which regularly pulls configuration catalogs from a Puppet master and applies them locally. This section includes how to sign the agent certificate request in the console.

These instructions assume that you've installed a monolithic PE deployment and have the Puppet master, the PE console, and PuppetDB up and running on one node.

**How does a Puppet agent work?**

Agents ensure that resources in a node stay in their desired state.

Periodically, a Puppet agent will send facts to a Puppet master and request a catalog. The master compiles the catalog using several sources of information, and returns the catalog to the agent.

Once 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, the agent will make any changes necessary to correct them. (Or, in no-op mode, it will report on what changes would have been needed.)

After applying the catalog, the agent submits a report to its master. Reports from all the agents are stored in PuppetDB and can be accessed in the PE console.
**Step 1a: Install an agent with the same OS and architecture as the Puppet master**

*Follow these steps if your agent node and your Puppet master have the same OS and architecture. Otherwise, go to Step 1b.*

1. Log into your agent node and run:

   ```
   curl -k https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash
   ```

   This script detects the agent's operating system, sets up an apt, yum, or zipper repo that refers back to the Puppet master, and then pulls down and installs the `puppet-agent` packages.

2. After installation is complete, approve the certificate request to approve the new agent's certificate request in the console.

**Step 1b: Install an agent with a different OS and architecture than the Puppet master**

*Follow these steps if your agent node and your Puppet master do not have the same OS and architecture. Otherwise, go to Step 1a.*

This example describes adding an agent running Debian 6 on AMD64 hardware. As you complete the steps, modify the commands to match your agent's OS and architecture.

1. In the console, click **Classification**, and in the **PE Infrastructure** node group, select the **PE Master** group.

2. On the **Configuration** tab, in the **Add new class** field, enter `pe_repo`, and select the class `pe_repo::platform::debian_6_amd64` — or whichever class matches your agent — from the list of classes.

   The repo classes are listed as `pe_repo::platform::<agent_os_version_architecture>`

3. Click **Add class**.

   The class you selected now appears in the list of classes for the **PE Master** group, but it has not yet been configured on your nodes. For that to happen, you need to kick off a Puppet run.

4. From the command line of your master, run `puppet agent -t`.

5. SSH into your agent node, and run the following:

   ```
   curl -k https://<master.example.com>:8140/packages/current/install.bash | sudo bash
   ```

   If you want to install a version of PE other than the most recent release, replace `current` in the script with a specific PE version number, in the form of `2016.x.x`.

   Depending on your platform, the method for downloading the script might vary.

   The installer installs and configures the Puppet Enterprise agent.

6. After installation is complete, approve the certificate request to approve the new agent's certificate request in the console.

**Step 2: Approve the certificate request**

During installation, the agent node contacts the Puppet master and requests a certificate. To add the node to the console and to start managing its configuration, you must approve its certificate request.

1. In the console, load a list of currently pending node requests by clicking **Unsigned certs**.

2. Click the **Accept All** button to approve the request and add the node.

The Puppet agent can now retrieve configurations from the master the next time Puppet runs.
**Step 3: Test the Puppet agent node**

You can wait until Puppet runs automatically, or you can manually trigger Puppet runs.

By default, the agent fetches configurations from the Puppet master every 30 minutes. (You can configure this interval in the puppet.conf file with the runinterval setting.) However, you can manually trigger a Puppet run from the command line at any time.

1. On the agent, log in as root and run `puppet agent --test`. This triggers a single Puppet run on the agent with verbose logging.

   If you receive a `bash: puppet: command not found` error, then the directory that PE installs its binaries in, `/opt/puppetlabs/bin`, isn’t included in your default `$PATH`. To include these binaries in your default `$PATH`, add them by running `PATH=/opt/puppetlabs/bin:$PATH;export PATH`.

2. Note the long string of log messages, ending with this message: `Notice: Applied catalog in [...] seconds`.

   You are now fully managing the agent node! It has checked in with the Puppet master for the first time and received its configuration info. It will continue to check in and fetch new configurations every 30 minutes.

Next, you'll begin learning how to configure your agents with Puppet code, beginning with pre-built chunks of Puppet code called **modules**.

**Start installing modules**

Modules are self-contained bundles of code and data. You can write your own modules, and you can download pre-built modules from the Puppet Forge. In this guide, you'll install a Puppet module, the primary means by which PE configures and manages nodes.

While you can use any module available on the Forge, PE customers can take advantage of supported modules. These modules are designed to make common services easier, and are tested and maintained by Puppet. In this getting started guide, you'll install the puppetlabs-apache module, a Puppet Enterprise-supported module.

These instructions assume you've installed a monolithic PE deployment, and have installed at least one *nix agent node.

**Step 1: Install a Forge module**

The Puppet Forge contains thousands of modules that you can use in your own environment.

1. Install the Apache module by running `puppet module install puppetlabs-apache`.

   The results look like this:

   ```
   Preparing to install into /etc/puppetlabs/code/environments/production/modules ...
   Notice: Downloading from http://forgeapi.puppetlabs.com ...
   Notice: Installing -- do not interrupt ...
   /etc/puppetlabs/code/environments/production/modules
   ### puppetlabs-apache (v1.1.1)
   ```

2. Visit the Puppet Forge to learn more about the **apache module**.

You have just installed a Puppet module! Modules contain classes, which are named chunks of Puppet code and are the primary means by which Puppet Enterprise configures and manages nodes.

Next, you'll use the **apache class** included in **puppetlabs-apache** to manage Apache on your agent node with Puppet Enterprise.
Start adding classes
In this guide, you’ll create a new node group, then apply a named chunk of Puppet code --- better known as a class --- to the node group. You’ll apply the apache class to your agent node, which will allow you to launch the default Apache virtual host on your agent node.

Complete a basic classification workflow
Classifying nodes associates classes with nodes so that the code they contain can be used by that node.

Before you begin
These instructions assume you’ve installed a monolithic PE deployment, and have installed at least one *nix agent node and the puppetlabs-apache module.

Every module contains one or more classes -- a named chunk of Puppet code. Classes are the primary means by which Puppet Enterprise configures nodes. The puppetlabs-apache module you installed in the module installation getting started guide contains a class called apache. In this section, you apply the apache class to your agent node

Create a node group
Node groups enable you to assign classes to more than one node at a time.

You could assign classes to individual nodes one at a time, but chances are, each of your classes needs to be applied to more than one node. By creating a node group, you can apply a class to many nodes at once.

1. In the console, click Classification, and click Add group.
2. Specify options for the new node group:
   - Parent name: select All nodes
   - Group name: enter a name that describes the role of this environment node group
   - Environment: select production
   - Environment group: don’t select this option
3. Click Add

Step 2: Add nodes to the node group
Add nodes to a node group to manage them more efficiently.

To add nodes to a node group, create rules that define which nodes should be included in the group.

1. Click the apache_example group.
2. From the Rules tab, in the Certname area, in the Node name field, enter the name of your PE agent node.
3. Click Pin node.
4. Commit changes.
5. Repeat these steps for any other nodes you want to add to the node group.

Step 3: Add the apache class to the node group
After you create a node group, add classes to give the matching nodes purpose.

1. Unless you have navigated elsewhere in the console, the apache_example node group should still be displayed in the Classification page. On the Configuration tab, in the Class name field, begin typing apache, and select it from the autocomplete list.
2. Click Add class, and commit changes.
   The apache class now appears in the list of classes for your agent. You can see this list by clicking Inventory and then clicking your node in the Inventory list. When the page opens with your node’s details, click the Configuration tab.
3. Apply your changes. From the command line on your agent, run puppet agent -t.
4. To see your changes in action, navigate to /var/www/html/, and create a file named index.html.
5. Open index.html with the text editor of your choice and add some content (for example, ”Hello, World!”).
6. From the command line of your agent node, run `puppet agent -t`. This configures the node using the newly assigned class.

7. Wait one or two minutes.

8. Open a browser and enter the IP address for the agent node, adding port 80 on the end, as in `http://myagentnodeIP:80/`. The contents of `/var/www/html/index.html` are displayed.

**Step 4: Edit class parameters in the console**

You can use the console to modify the values of a class's parameters without editing the module code directly.

1. In the console, click Classification, and then find and select the `apache_example`.

2. On the Configuration tab, find `apache` in the list of classes.

3. From the Parameter name drop-down list, choose the parameter you want to edit. For this example, select `docroot`.

   **Note:** The gray text that appears as values for some parameters is the default value, which can be either a literal value or a Puppet variable. You can restore this value by selecting Discard changes after you have added the parameter.

4. In the Value field, enter `/var/www`.

5. Click Add parameter, and commit changes.

6. From the command line of your PE-managed node, run `puppet agent -t`.

   This triggers a Puppet run, and Puppet Enterprise creates the new configuration.

   You have set the Apache web server's root directory to `/var/www` instead of its default `/var/www/html`. If you refresh `http://myagentnodeIP:80/` in your web browser, it shows the list of files in `/var/www`. If you click html, the browser again shows the contents of `/var/www/html/index.html`.

   Puppet Enterprise is now managing the default Apache vhost on your agent node. Next, you'll learn how to manage users with PE, and how to set permissions for each user or group of users.

**Start assigning user permissions**

This section introduces you to role-based access control. You’ll learn how to manage users by creating a user role, granting that role permissions to work with a node group, and assigning users to the role.

Create a new user role and give the role the permission to view the node group you created in the Adding classes section. In addition, create a new local user, and assign your new user role to that user.

PE enables you to create and manage users and user groups through the console. You can also create user roles, and assign users to those roles. Permissions, such as the ability to view node groups, deploy code, or generate password reset tokens, are assigned to user roles rather than directly to users. When you assign roles to users or user groups, you are granting permissions in a more organized way.

**Complete a basic RBAC workflow**

Role-based access control (RBAC) is used to manage user permissions. Permissions define what actions users can perform on designated objects. There are multiple steps involved in an RBAC workflow, which can be adapted to fit your needs.

**Before you begin**

These instructions assume you’ve installed a monolithic PE deployment, and have installed at least one *nix agent node and the puppetlabs-apache module.

**Note:** Roles are deletable by API, not in the console. Therefore, it's best to try out these steps on a virtual machine.

**Create a user role**

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.

1. In the console, click User roles.
2. For **Name**, type **Web developers**, and then for **Description**, type a description for the Web developers role, such as **Members of the web development team**.

3. Click **Add role**.

**Step 2: Give the user role access to a node group**

Users get permissions when the user roles they belong to are granted those permissions.

One of the permissions you can grant to user roles is the ability to access (view, create, or edit) node groups.

1. From the **User Roles** page, select **Web developers**, and then click the **Permissions tab**.
2. In the **Type** box, select **Node groups**.
3. In the **Permission** box, select **View**.
4. In the **Object** box, select **apache_example**.
5. Click **Add permission**, and then click the **commit** button.

You have given members of the **Web developers** role permission to view the **apache_example** node group.

**Step 3: Create a new user**

You can add local users, or import users and groups from a directory.

These steps add a local user. You can also import users and groups from an external directory, so you don't have to recreate users one at a time.

1. In the console, click **Users**.
2. In the **Full name** field, type in a user name.
3. In the **Login** field, type the user’s login information.
4. Click **Add local user**.

**Related information**

*Connecting external directory services to PE on page 283*

Puppet Enterprise connects to external Lightweight Directory Access Protocol (LDAP) directory services through its role-based access control (RBAC) service. This allows you to use existing users and user groups that have been set up in your external directory service.

**Step 4: Enable the new user to log in**

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. Click the new local user in the **Users** list.
2. On the upper-right of the page, click **Generate password reset**. A **Password reset link** message box opens.
3. Copy the link provided in the message and send it to the new user. Then you can close the message.

**Step 5: Assign the user role to the new user**

Users must be assigned to one or more roles before they can log in and use PE.

When you add users to a role, the user gains the permissions that are applied to that role.

1. Click **User Roles** and then click **Web developers**.
2. On the **Member users** tab, on the **User name** list, select the new user you created, and then click **Add user** and click the commit button.

You're now managing a user with RBAC. By using permissions and user roles, you give the appropriate level of access and agency to each user or user group who works with PE.

Next, you'll learn more about the basics of writing modules of your own, so you can begin customizing your deployment and getting work done.
Start writing modules for *nix

In this guide, you'll modify a Forge module, and you'll use the Puppet Development Kit to write a module and unit test it. This will help you become more familiar with Puppet modules and module development.

Modules are reusable chunks of Puppet code and are the basic building blocks of any Puppet Enterprise (PE) deployment. Some modules from the Forge will precisely fit your needs, but many are almost --- but not quite --- what you need. You'll sometimes want to adapt pre-written modules to suit your deployment's requirements. In other cases, you'll need to write your own modules from scratch.

**CAUTION:** The directories and some of the commands in this guide do not work correctly with . If you are using, do not follow the steps in this guide.

Module location

By default, Puppet keeps modules in /etc/puppetlabs/code/environments/production/modules. This includes modules installed by PE, those that you download from the Forge, and those you write yourself. You can configure this path with the modulepath setting in puppet.conf.

**Note:** PE also creates another module directory at /opt/puppetlabs/puppet/modules. Don't modify anything in this directory or add modules of your own to it.

Module structure

Modules are directory trees that contain manifests, templates, and other files.

You'll find the following files in the puppetlabs-apache module:

- apache/ (the module name)
  - manifests/
    - init.pp (contains the apache class)
    - php.pp (contains the php class to install PHP for Apache)
    - vhost.pp (contains the Apache virtual hosts class)
  - templates/
    - vhost/
      - _file_header.erb (contains the vhost template, managed by PE)

Every manifest (.pp) file contains a single class. File names map to class names in a predictable way:

- init.pp contains a class with the same name as the module, in this case apache.
- php.pp contains a class called <MODULE NAME>::<NAME>.
- vhost.pp contains a class called <MODULE NAME>::<NAME>::<OTHER NAME>.

Many modules, including Apache, contain directories other than manifests and templates. For simplicity's sake, we do not cover them in this introductory guide.

Edit a module's manifest and use the edited module

Modules from the Forge provide a great basis for common tasks. You can configure them to meet your specific needs.

Before you begin

Make sure you've installed:

- A monolithic PE deployment
- At least one *nix agent node
- the puppetlabs-apache module

In this simplified exercise, you modify a template from the puppetlabs-apache module, specifically vhost.conf.erb, to include some simple variables that will be automatically populated with facts about your node.

**Note:** You should be logged in as root or administrator on your nodes.

© 2020 Puppet, Inc.
1. On the Puppet master, navigate to the modules directory by running cd /etc/puppetlabs/code/environments/production/modules

2. Run ls to view the currently installed modules and note that apache is present.

3. Open apache/templates/vhost/_file_header.erb in a text editor. (Avoid using Notepad because it can introduce errors.) The _file_header.erb file contains the following header:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
# ************************************
```

4. Use the PE lookup tool, Facter, to collect the following facts about your agent node:
   - run facter operatingsystem (this returns your agent node’s OS)
   - run facter id (this returns the id of the currently logged in user)

5. Edit the header of _file_header.erb so that it contains the following variables for Facter lookups:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
# This file is authorized for deployment by <%= scope.lookupvar('::id') %>.  
# This file is authorized for deployment ONLY on  
<%= scope.lookupvar('::operatingsystem') %> <%= scope.lookupvar('::operatingsystemmajrelease') %>.  
# Deployment by any other user or on any other system is strictly prohibited.
# ************************************
```

6. In the console, add apache to the available classes, and then add that class to your agent node. Refer to the Adding classes getting started guide if you need help with these steps.

   When Puppet runs, it configures Apache and starts the httpd service. When this happens, a default Apache vhost is created based on the contents of _file_header.erb.

7. On the agent node, navigate to one of the following locations, depending on your operating system:
   - Redhat-based: /etc/httpd/conf.d
   - Debian-based: /etc/apache2/sites-available

8. View 15-default.conf; depending on the node’s OS, the header shows some variation of the following contents:

```
# ************************************
# Vhost template in module puppetlabs-apache
# Managed by Puppet
# This file is authorized for deployment by root.
# This file is authorized for deployment ONLY on Redhat 6.
# Deployment by any other user or on any other system is strictly prohibited.
# ************************************
```

As you can see, PE has used Facter to retrieve some key facts about your node, and then used those facts to populate the header of your vhost template.
Related information
Start adding classes on page 94
In this section, you'll use the console to add a class to your Puppet agent. Classes are named chunks of Puppet code that are stored in modules. The class you assign in this exercise is derived from the module you installed previously.

Write a new module
Write, validate, and test an example module for PE on *nix systems with Puppet Development Kit (PDK).

Before you begin
Make sure you've installed:
• A monolithic PE deployment.
• At least one *nix agent node.
• The puppetlabs-apache module.
• Puppet Development Kit on your Puppet master. See PDK installation instructions.
You must be logged in as root or administrator on your nodes.
Tip: PDK is a standalone development kit that doesn't require Puppet on your development machine. For simplicity in this example, you'll create your module on your master, but normally, you would develop modules on a separate development workstation and then move your module to the master.

Create a pe_getstarted_app module that manages a PHP-based web app running on an Apache virtual host.

1. Make sure you're in the modules directory by running cd /etc/puppetlabs/code/environments/production/modules
2. On the master, from the command line, run pdk new module pe_getstarted_app --skip-interview
By default, PDK asks a series of metadata questions before it creates your module. The --skip-interview option bypasses the interview, and PDK uses default values for the module's metadata.

PDK creates the new module’s directory with the same name you gave the module. It also creates the module's metadata, subdirectories, and testing template files.
3. Change into the module directory by running cd pe_getstarted_app
4. Validate your module's code syntax and style by running pdk validate
The pdk validate command checks the Puppet and Ruby code style and syntax in your module. Validate every time you add new code.
5. Unit test your module by running pdk test unit to ensure that the testing directories and templates were correctly created.
On a newly generated module, the pdk test unit command checks that the testing directories and templates were correctly created. As you add new code to your module, you'll write and run unit tests to make sure your code works.

Create a module class
Write, test, and deploy a module class for an example module.

Before you begin
Make sure you have generated the pe_getstarted_app module as described in the Write a module topic.

1. In the pe_getstarted_app directory, create the main class for your module by running pdk new class pe_getstarted_app
When you create a class with the same name as your module, PDK creates the init.pp file. This file contains the main class of a module, and it's the only class with a file name that's different from the class name. PDK also creates testing directories and templates for the class.
2. Validate your class's code syntax and style by running pdk validate

© 2020 Puppet, Inc.
3. Unit test your class by running `pdk test unit`

4. Open the `init.pp` file in your text editor and add the following Puppet code to it. Save the file and exit the editor.

```puppet
class pe_getstarted_app (
    $content = "<?php phpinfo() ?>\n",
) {

    class { 'apache':
        mpm_module => 'prefork',
    }

    include apache::mod::php

    apache::vhost { 'pe_getstarted_app':
        port => '80',
        docroot => '/var/www/pe_getstarted_app',
        priority => '10',
    }

    file { '/var/www/pe_getstarted_app/index.php':
        ensure => file,
        content => $content,
        mode => '0644',
    }
}
```

Additional details about the code in your new class:

- The class `apache` is modified to include the `mpm_module` attribute. This attribute determines which multi-process module is configured and loaded for the Apache (HTTPD) process. In this case, the value is set to 'prefork'.
- `include apache::mod::php` indicates that your new class relies on those classes to function correctly. PE understands that your node needs to be classified with these classes and completes that work automatically when you classify your node with the `pe_getstarted_app` class; in other words, you don’t need to worry about classifying your nodes with Apache and Apache PHP.
- The priority attribute of '10' ensures that your app has a higher priority on port 80 than the default Apache vhost app.
- The file `/var/pe_getstarted_app/index.php` contains whatever is specified in the `content` attribute. This is the content you see when you launch your app. PE uses the `ensure` attribute to create that file the first time the class is applied.

After creating your class, you are ready to validate and unit test it.

**Validate and unit test a class**

Validate and unit test a class in an example module.

When you add code to a module, you should always validate and unit test it. PDK creates unit test directories and templates, and then you write the unit tests you need. For this example, we've provided an rspec unit test. To learn more about how to write unit tests, see rspec-puppet documentation.

1. In the `pe_getstarted_app` directory, create a `.fixtures.yml` file. This file installs module dependencies for unit testing.
2. Open the `.fixtures.yml` file and edit it to include the following code. Save the file and exit the editor.

```yaml
fixtures:
  forge_modules:
    apache: "puppetlabs/apache"
    stdlib: "puppetlabs/stdlib"
    concat: "puppetlabs/concat"
```

3. Change into the spec tests directory by running `cd spec/classes`.
4. Open the `pe_getstarted_app_spec.rb` in your text editor and add the following code. Save and exit the file.

```ruby
require 'spec_helper'

describe 'pe_getstarted_app', type: :class do
  let!(:facts) do
    { 
      operating_system_release: '14.04',
      osfamily: 'Debian',
      operating_system: 'Ubuntu',
      lsb_dist_release: 'Trusty',
    }
  end

  describe 'standard content' do
    it { is_expected.to contain_class('apache').with('mpm_module' => 'prefork') }

    it { is_expected.to contain_apache__vhost('pe_getstarted_app').with(
      'port' => '80',
    )
    }

    it { is_expected.to contain_file('/var/www/pe_getstarted_app/
      index.php').with(
      'ensure' => 'file',
      'content' => "<?php phpinfo() ?>\n",
      'mode' => '0644'
    )
  end

  describe 'custom content' do
    let!(:params) do
      { 'content' => "custom\n", }
    end

    it { is_expected.to contain_file('/var/www/pe_getstarted_app/
      index.php').with(
      'ensure' => 'file',
      'content' => "custom\n",
      'mode' => '0644'
    )
  end
end
```

5. Run `pdk validate` to check the code style and syntax of your class.
6. Run `pdk test unit` to run your unit test against your class.

**Adding your module class to nodes**

Add the class from your example module to nodes in the console.

You can now add your new module's class to the console and apply it to nodes, following the workflow in the Adding classes getting started guide.

Congratulations! Your first module is up and running. There are plenty of additional resources about modules and the creation of modules that you can reference. Check out documentation about Puppet Development Kit, module fundamentals, the modulepath, the Beginner's guide to modules, and the Puppet Forge.

**Related information**

- [Start adding classes](#) on page 94

In this section, you'll use the console to add a class to your Puppet agent. Classes are named chunks of Puppet code that are stored in modules. The class you assign in this exercise is derived from the module you installed previously.

**Next steps**

After you've worked through the Puppet Enterprise (PE) getting started steps, you can now perform the core workflows of a Puppet user.

To continue learning, take the following next steps:

- Learn more about working with the [Forge](#) to find the modules you need.
- Use the [roles and profiles](#) method to create a complete system configuration and store it in a control repository.
- Manage Puppet code using [PE code management](#) tools and a control repository.
- Classify nodes using [groups and rules](#).

**Getting started on Windows**

Welcome to Puppet Enterprise (PE) getting started for Windows users. These pages demonstrate some essential tasks to start using PE, either to help you set up an initial installation for actual deployment, or as a way for you to try out some fundamentals of configuration management.

You'll walk through the setup of a monolithic installation: a PE deployment in which the Puppet master, the PE console, and PuppetDB are all installed on one node. The various sections of the getting started guide show you how to automate some basic tasks that sysadmins regularly perform.

**Important:** For the most part, interacting with Puppet is the same regardless of your operating system. The key difference between other operating systems and Windows is that you cannot configure a Windows machine to be a Puppet master. Agent components can be installed on Windows machines and you can manage those machines with your Linux master.

The exercises build on each other to lay in some foundational configuration tasks. Work through them in the order presented. See the prerequisite sections in each page to ensure you have the correct setup to perform the steps as they're provided.

- [Start installing PE in a Windows environment](#) on page 89

This getting started guide uses a web-based installer to walk you through a basic monolithic Puppet Enterprise (PE) installation, ideal for trying out PE with up to 10 nodes, or for managing up to 4000 nodes.

- [Start installing Windows agents](#) on page 92

When a node is managed by Puppet, it runs a Puppet agent application, commonly called an agent. In this section you'll install a Windows Puppet agent, which regularly pulls configuration catalogs from a Puppet master and applies them locally. These instructions include how to sign the agent certificate request in the console.
• Start installing modules on page 93
  Modules are shareable, reusable units of Puppet code that extend Puppet across your infrastructure by automating tasks such as setting up a database, web server, or mail server. In this section, you’ll install a module from the Puppet Forge.

• Start adding classes on page 94
  In this section, you’ll use the console to add a class to your Puppet agent. Classes are named chunks of Puppet code that are stored in modules. The class you assign in this exercise is derived from the module you installed previously.

• Start assigning user access on page 95
  The console enables you to import users and groups, create user roles, and assign users to roles. In this exercise, you create a user role, and give the role view permissions on the node group you previously created. Then you create a local user, and assign a user role to that user.

• Start writing modules for Windows on page 96
  In this section, you’ll learn about Puppet modules and module development by writing your own basic module. You’ll create a site module and use the console to apply your new module’s class to a group.

• Next Steps on page 99
  After you’ve worked through the Puppet Enterprise (PE) getting started steps, you can now perform the core workflows of a Puppet user.

Start installing PE in a Windows environment

This getting started guide uses a web-based installer to walk you through a basic monolithic Puppet Enterprise (PE) installation, ideal for trying out PE with up to 10 nodes, or for managing up to 4000 nodes.

**Important:** The Puppet master can only run on *nix machines, but Windows machines can run as Puppet agents and you can manage those nodes with your *nix Puppet master. This getting started guide assumes you want to access the master *nix machine remotely from a Windows machine. If you plan to install directly onto a *nix node, follow the installing PE instructions in the *nix getting started guide.

Here we use a web-based installer to walk you through a basic monolithic Puppet Enterprise installation, ideal for trying out PE with up to 10 nodes. On a single *nix machine, you’ll install:

• The Puppet master, the central hub of activity, where Puppet code is compiled to create agent catalogs, and where SSL certificates are verified and signed.
• The console, PE's web interface, which features numerous configuration and reporting tools.
• PuppetDB, which collects data generated throughout your Puppet infrastructure.

**Step 1: Review installation prerequisites on your Linux server**

Before getting started, review this checklist to make sure you're ready to install PE.

**Note:** The examples in this guide use a Linux server running Red Hat Enterprise Linux (RHEL) 6.

1. Be aware that you'll need to work as the root user on the command line throughout the installation process.
2. Make sure you meet the hardware recommendations for 10 or fewer nodes.
   You can download and install Puppet Enterprise on up to 10 nodes at no charge.
3. Make sure that DNS is properly configured on the server you're installing on.
   • All nodes must know their own hostnames, which you can achieve by properly configuring reverse DNS on your local DNS server, or by setting the hostname explicitly. Setting the hostname usually involves the hostname command and one or more configuration files, but the exact method varies by platform.
   • All nodes must be able to reach each other by name, which you can achieve with a local DNS server.
4. Know the fully qualified domain name (FQDN) of the server you're installing PE on, for example, master.example.com.
5. The web-based installer requires access to port 3000 on the server you're running the installer from.

**Related information**

Hardware requirements on page 148

© 2020 Puppet, Inc.
These hardware requirements are based on internal testing at Puppet and are meant only as guidelines to help you determine your hardware needs.

**Step 2: Prepare your Windows System**
Some extra software is necessary so that you can communicate and work between your Windows and Linux machines.

On your Windows machine:
1. Set up an SFTP client. Popular clients include FileZilla and WinSCP. The examples below use WinSCP.
2. Set up an SSH Client. The examples below use the Putty SSH client.

**Step 3: Install the master on your Linux machine**
These steps show you how to use a Windows machine to install a Puppet master on a Linux server.

1. Open the SSH client, and enter the IP address and port of the Linux machine that you want to use as your master. You can then open an SSH session to the master.
2. When prompted in the terminal, log into the Linux node as the root user.
3. Download the PE installer to the server that will be your master by copying the appropriate download URL from the downloads page and running:
   
   ```bash
   wget --content-disposition '<DOWNLOAD_URL>'
   ```
4. The web-based installer requires access to port 3000 on the node you're running the installer from. If you can't connect directly to port 3000, you can port forward — or "tunnel" — to the installer using SSH.
   a) Open PuTTY. Select Sessions and in the Host Name field, enter the FQDN of the host you want to run the installer from.
   b) Select Tunnels and in the Source Port field, enter 3000.
   c) In the Destination field, enter localhost:3000.
   d) Select Local, click Add, and then click Open.
5. Run the following command to unpack the tarball:
   
   ```bash
   tar -xf <TARBALL>
   ```
   
   **Note:** You need about 1 GB of space in `/tmp` to untar the installer.
6. Change directories to the installer directory, and run the installer:
   
   ```bash
   sudo ./puppet-enterprise-installer
   ```

© 2020 Puppet, Inc.
7. When prompted, choose the guided installation option.
   
   The installer starts a web server and provides an installer URL.
   
   The default installer URL is `https://<INSTALL_PLATFORM_HOSTNAME>:3000`. If you forwarded ports in step 1, the URL is `https://localhost:3000`.
   
   Copy this address into your browser (make sure to use `https`).

8. In a browser, access the installer URL and accept the security request.
   
   The installer uses a default SSL certificate. You must add a security exception in order to access the installer.
   
   **Important:** Leave your terminal connection open until the installation is complete or else installation fails.

9. On the installer page, click **Let's get started**.

![Installer URL](image)

**Note:** From this point the examples will refer to your Linux server as the master.

**Related information**

[Accepting the console's certificate](#) on page 274

The console uses an SSL certificate created by your own local Puppet certificate authority. Since this authority is specific to your site, web browsers won't know it or trust it, and you'll have to add a security exception in order to access the console.

**Step 4: Personalize your installation**

The installer collects key configuration information, which is used to make sure your installation is set up correctly.

Follow the prompts to configure your installation:

1. Choose **Install on this server**, which will install the Puppet master component on the Linux server.

2. Provide the following information about the Puppet master server:
   
   - Puppet master FQDN: The fully qualified domain name of the server you're installing PE on.
   - DNS aliases: A comma-separated list of **DNS alt names** or aliases that your agent nodes (the nodes you wish to manage with PE) can use to reach the Puppet master.

3. When prompted about database support, choose the default option, **Install PostgreSQL for me**.

4. Create a password for the console administrator. The password must be at least eight characters. The user name for the console administrator is always `admin`.

5. Click **Submit**, and then review the information you provided. If you need to make any changes, click **Go back** and make whatever changes are required. Otherwise, click **Continue**.

**Log into the PE console**

On the validation page, the installer verifies various configuration elements (for example, if SSH credentials are correct, if there is enough disk space, and if the OS is the same for the various components).
1. When you click **Start using Puppet Enterprise**, you receive a browser warning about an untrusted certificate. This is 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 login page for the console, log in with the username `admin` and the password you created when installing the Puppet master.

Congratulations, you've successfully installed the Puppet master node! Next you'll install PE on your agent nodes, so that you can manage these nodes with Puppet.

### Start installing Windows agents

When a node is managed by Puppet, it runs a Puppet agent application, commonly called an agent. In this section you'll install a Windows Puppet agent, which regularly pulls configuration catalogs from a Puppet master and applies them locally. These instructions include how to sign the agent certificate request in the console.

These instructions assume that you've installed a monolithic PE deployment and have the Puppet master, the PE console, and PuppetDB up and running on one node.

#### How does a Puppet agent work?

Agents ensure that resources in a node stay in their desired state.

Periodically, a Puppet agent will send facts to a Puppet master and request a catalog. The master compiles the catalog using several sources of information, and returns the catalog to the agent.

Once 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, the agent will make any changes necessary to correct them. (Or, in no-op mode, it will report on what changes would have been needed.)

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

### Step 1: Install an agent on your Windows machine

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.

You must use PowerShell 2.0 or later to install Windows agents with PE package management.

**Note:** The `<MASTER HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the master. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.

2. On the **Configuration** tab in the **Class name** field, select `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 master, run Puppet to configure the newly assigned class.

   The new repository is created on the master at `/opt/puppetlabs/server/data/packages/public/ <PE VERSION>/<PLATFORM>/`.

5. On the node, open an administrative PowerShell window, and install:

   ```powershell
   ```
After running the installer, the following output indicates the agent was successfully installed.

Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'
   service { 'puppet': ensure => 'running', enable => 'true', }

**Step 2: Approve the certificate request**

During installation, the agent node contacts the Puppet master and requests a certificate. To add the node to the console and to start managing its configuration, you must approve its certificate request.

1. In the console, load a list of currently pending node requests by clicking **Unsigned certs**.
2. Click the **Accept All** button to approve the request and add the node.

The Puppet agent can now retrieve configurations from the master the next time Puppet runs.

**Step 3: Test the Windows agent node**

You can wait until Puppet runs automatically, or you can manually trigger Puppet runs.

By default, the agent fetches configurations from the Puppet master every 30 minutes. You can configure this interval in the `puppet.conf` file with the `runinterval` setting. Alternatively, you can manually trigger a Puppet run from the command line at any time.

To start a Puppet run, run `puppet agent -t` on the SSH client.

This triggers a single Puppet run on the agent with verbose logging.

Note the long string of log messages, ending with this message:

Notice: Applied catalog in <N> seconds.

You are now fully managing the agent node! It has checked in with the Puppet master for the first time and received its configuration info. It will continue to check in and fetch new configurations every 30 minutes.

Next, you’ll begin learning how to configure your agents with Puppet code, beginning with pre-built chunks of Puppet code called modules.

### Start installing modules

Modules are shareable, reusable units of Puppet code that extend Puppet across your infrastructure by automating tasks such as setting up a database, web server, or mail server. In this section, you’ll install a module from the Puppet Forge.

The Puppet Forge is a repository for modules created by Puppet and the Puppet community. It contains thousands of modules submitted by users and Puppet developers for you to use. In addition, PE customers can take advantage of supported modules, which are designed to make common services easier and are tested and maintained by Puppet.

In this section, you’ll install the `puppetlabs-wsus_client` module, a Puppet Enterprise supported module. In a subsequent section, **Writing modules for Windows**, you’ll learn more about modules, including how to write your own.

The process for installing a module is the same on both Windows and *nix operating systems.

These instructions assume you have installed a monolithic PE deployment, and installed at least one Windows agent node.

**Step 1: Create the modules directory**

By default, Puppet keeps modules in `C:\ProgramData\PuppetLabs\code\environments\production\modules` or, on *nix `/etc/puppetlabs/code/environments/production/modules`. This includes modules installed by PE, those that you download from the Forge, and those you write yourself. In a fresh installation, you need to create this modules subdirectory yourself.

**Note:** PE creates two other module directories: `/opt/puppetlabs/puppet/modules` and `/etc/puppetlabs/staging-code/modules`. Don’t modify anything in or add modules of your own to `/opt/`
puppetlabs/puppet/modules. The /etc/puppetlabs/staging-code/modules directory is for file sync use only; if you are not using Code Manager or file sync, do not add code to this directory.

Navigate to the production directory and run: mkdir modules

**Step 2: Install a Forge module**
The wsus_client module, or Windows Server Update Service (WSUS) module, lets Windows administrators manage operating system updates using their own servers instead of Microsoft's Windows Update servers.

Run puppet module install puppetlabs-wsus_client

If you're installing a module directly on your Windows machine, the output looks like this:

```
PS C:\Users\Administrator> puppet module install puppetlabs-wsus_client
Notice: Preparing to install into
C:/ProgramData/PuppetLabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
C:/ProgramData/PuppetLabs/code/environments/production/modules
+- puppetlabs-wsus_client (v1.0.1)
  +- puppetlabs-registry (v1.1.3)
  +- puppetlabs-stdlib (v4.11.0)
```

This is the output (on a *nix machine):

```
Preparing to install into /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from http://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
### puppetlabs-wsus_client (v1.0.1)
### puppetlabs-registry (v1.1.3)
### puppetlabs-stdlib (v4.11.0)
```

You have just installed a module. You can read about this module in the module readme. All of the classes in the module are now available to be added to the console and assigned to nodes.

Next: Adding classes to Windows agent nodes.

**Start adding classes**

In this section, you'll use the console to add a class to your Puppet agent. Classes are named chunks of Puppet code that are stored in modules. The class you assign in this exercise is derived from the module you installed previously.

The Puppet wsus_client module contains a class called wsus_client. In this exercise, you'll use the wsus_client class to supply the types and providers necessary to schedule updates using a WSUS server with Puppet.

To prepare the class, you create a group called windows_example and add the wsus_client class to it.

**Note:** The process for adding classes to agent nodes in the console is the same on both Windows and *nix operating systems.

**Step 1: Create the windows_example group**

To classify a node is to add classification data (classes, parameters, and variables) for the node group to the node.

**Before you begin**

These instructions assume you have installed a monolithic PE deployment, installed at least one Windows agent node, and installed the puppetlabs-wsus_client module.

1. In the console, click Classification, and click Add group.
2. In the Group name field, name your group, for example, windows_example, and click Add.
3. Click **Add membership rules, classes, and variables**.
4. On the **Rules** tab, in the **Certname** field, enter the name of the managed node you want to add to this group, and click **Pin node**.

   Repeat this step for any additional nodes you want to add.

   **Note**: Pinning a node adds the node to the group regardless of any rules specified for a node group. A pinned node remains in the node group until you manually remove it. Adding nodes dynamically describes how to use rules to add nodes to a node group.

5. Commit your changes.

**Step 2: Add the **wsus_client** class to the example group**

Adding the class to the group and then running Puppet configures the group to use that class.

1. In the console, click **Classification**, and find and select the **windows_example** group.
2. On the **Configuration** tab, in the **Add new class** field, select **wsus_client**.

   If **wsus_client** doesn't appear in the list, you might have to click **Refresh**.
3. Click **Add class**, and commit changes.

   The **wsus_client** class now appears in the list of classes for your agent node.

4. Puppet runs, which configures the **windows_example** group using the newly-assigned class. Wait one or two minutes.

**Start assigning user access**

The console enables you to import users and groups, create user roles, and assign users to roles. In this exercise, you create a user role, and give the role view permissions on the node group you previously created. Then you create a local user, and assign a user role to that user.

You can connect Puppet Enterprise (PE) with an external directory, such as Active Directory or OpenLDAP, and import users and groups, rather than creating and maintaining users and groups in multiple locations. You can create user roles, and assign imported users to those roles. Roles are granted permissions, such as permission to act on node groups. When you assign roles to users or user groups, you are granting users permissions in a more organized way.

This exercise doesn't cover connecting with an OpenLDAP or Active Directory.

**Note**: Users and user groups are not currently deletable. And roles are deletable by API, not in the console. Therefore, we recommend that you try out these steps on a virtual machine.

**Step 1: Create a user role**

Add a user role so you can manage permissions for groups of users at one time.

**Before you begin**

Ensure you have installed a monolithic PE deployment, installed at least one Windows agent node, installed the **puppetlabs-wsus_client** module, and classified a node.

You must have admin permissions to complete these steps, which include assigning a user to a role.

1. In the console, click **User Roles**.
2. For **Name**, type **Windows users**, and then for **Description**, type a description for the role, such as **Windows users**.
3. Click **Add role**.

**Step 2: Create a user and add the user to your role**

These steps demonstrate how to create a new local user.

1. In the console, click **Users**.
2. In the **Full name** field, type in a user name.
3. In the **Login** field, type a username for the user.
4. Click **Add local user**.

   **Note:** When you create new local users, you need to send them a login token. Do this by clicking the new user's name in the **User list** and then on the upper-right of the user's page, click **Generate password reset**. A message opens with a link that you must copy and send to the new user.

5. Click **User Roles** and then click **users**.
6. On the **Member** users tab, on the **User name** list, select the new user you created, and then click **Add user** and click the **Commit** button.

**Step 3: Enable a user to log in**
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 **Users** page, click the new local user. The new user's page opens.
2. On the upper-right of the page, click **Generate password reset**. A **Password reset link** message box opens.
3. Copy the link provided in the message and send it to the new user. Then you can close the message.

**Step 4: Give your role access to the node group you created**
You must give the role access to the group, so that the **Windows users** role can view the **windows_example** node group.

1. From the **Windows users role** page, click the **Permissions** tab.
2. In the **Type** box, select **Node groups**.
3. In the **Permission** box, select **View**.
4. In the **Object** box, select **windows_example**.
5. Click **Add permission**, and then click the commit button.

**Start writing modules for Windows**
In this section, you'll learn about Puppet modules and module development by writing your own basic module. You'll create a site module and use the console to apply your new module's class to a group.

**Note:** This guide assumes that you are not using r10k for Code Manager. If you are using r10k, any modules not managed by r10k will be destroyed.

**Module basics**
Modules are directory trees. Many modules contain more than one directory.

By default, modules are stored in `/etc/puppetlabs/code/environments/production/modules`. If you're working from a Windows machine, the path is, `C:\ProgramData\PuppetLabs\code\environments\production\modules`. You can configure this path with the `modulepath` setting in `puppet.conf`.

The manifest directory of the Puppet `wsus_client` module contains the following files:

- `wsus_client/` *(the module name)*
  - `manifests/`
    - `init.pp` *(contains the `wsus_client` class)*
    - `service.pp` *(defines `wsus_client::service`)*

Every manifest (.pp) file contains a single class. File names map to class names in a predictable way: `init.pp` contains a class with the same name as the module; `<NAME>.pp` contains a class called `<MODULE NAME>::<NAME>;` and `<NAME>/<OTHER NAME>.pp` contains `<MODULE NAME>::<NAME>::<OTHER NAME>`. 

© 2020 Puppet, Inc.
Many modules contain directories other than manifests; for simplicity's sake, we do not cover them in this introductory section.

**Write a Puppet module**

Puppet modules save time, but at some point most users also need to write their own modules.

These instructions assume you have completed all of the preceding sections in Getting started with Puppet Enterprise for Windows users.

**Step 1: Write a class in a module**

Follow these steps to create a module with a single class called `critical_policy` that manages a collection of important settings and options in your Windows registry, most notably the legal caption and text users see before the login screen.

The new class has these characteristics:

- The `registry::value` defined resource type allows you to use Puppet to manage the parent key for a particular value automatically.
- The `key` parameter specifies the path the key the values must be in.
- The `value` parameter lists the name of the registry values to manage. This is copied from the resource title if not specified.
- The `type` parameter determines the type of the registry values. Defaults to 'string'. Valid values are 'string', 'array', 'dword', 'qword', 'binary', or 'expand'.
- `data` Lists the data inside the registry value.

1. On the Puppet master, make sure you're still in the modules directory, `cd /etc/puppetlabs/code/environments/production/modules`, and then run `mkdir -p critical_policy/manifests` to create the new module directory and its manifests directory.

2. Use your text editor to create and open the `critical_policy/manifests/init.pp` file.

3. Edit the `init.pp` file so it contains the following Puppet code, and then save it and exit the editor:

```puppet
class critical_policy {
  registry::value { 'Legal notice caption':
    key   => 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System',
    value => 'legalnoticecaption',
    data  => 'Legal Notice',
  }

  registry::value { 'Legal notice text':
    key   => 'HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System',
    value => 'legalnoticetext',
    data  => 'Login constitutes acceptance of the End User Agreement',
  }
}
```

For more information about writing classes, refer to the following documentation:

- To learn how to write resource declarations, conditionals, and classes in a guided tour format, start at the beginning of Learning Puppet.
- For a complete but succinct guide to the Puppet language's syntax, see the Puppet language reference.
- For complete documentation of the available resource types, see the type reference.
- For short, printable references, see the modules cheat sheet and the core types cheat sheet.
Step 2: Use your custom module in the console
Puppet recognizes when you create a custom class, and it can be added to the console and assigned to your Windows nodes.

1. In the console, click Classification, and select the node group you want to add your module to (for example, Windows_example).
2. On the Configuration tab, in the Add new class field, enter critical_policy.
   You might need to wait a moment or two for the class to show up in the list. You can also click Refresh.
3. Click Add class, and commit changes.

Step 3: Test out your module
Check to make sure your module does what you expect it to: display the legal caption and text before you log in.

1. On the Windows agent node, manually set the data values of legalnoticecaption and legalnoticetext to some other values. For example, set legalnoticecaption to “Larry’s Computer” and set legalnoticetext to “This is Larry’s computer.”
2. On the Windows agent node, refresh the registry and note that the values of legalnoticecaption and legalnoticetext have been returned to the values specified in your critical_policy manifest.
3. Reboot your Windows machine to see the legal caption and text before you log in again. You have created a new class from scratch and used it to manage registry settings on your Windows server.

Step 4: Use a site module
Many users create a "site" module for a type of machine.

Instead of describing smaller units of a configuration, the classes in a site module describe a complete configuration for a given type of machine. For example, a site module might contain:

- A site::basic class, for nodes that require security management but haven’t been given a specialized role yet.
- A site::webserver class for nodes that serve web content.
- A site::dbserver class for nodes that provide a database server to other applications.

Site modules hide complexity so you can more easily divide labor at your site. System architects can create the site classes, and junior admins can create new machines and assign a single "role" class to them in the console. In this workflow, the console controls policy, not fine-grained implementation.

1. On the Puppet master, create the /etc/puppetlabs/code/environments/production/modules/site/manifets/basic.pp file by running mkdir -p site/manifets. Then, edit it to contain the following:

   ```
   class site::basic {
     if $osfamily == 'windows' {
       include critical_policy
     }
     else {
       include motd
       include core_permissions
     }
   }
   ```

2. Run puppet agent -t to ensure site::basic is created. This class declares other classes with the include function.

   **Note:** The "if" conditional sets different classes for different OSs using the $osfamily fact. In this example, if an agent node is not a Windows agent, Puppet applies the motd and core_permissions classes. For more information about declaring classes, see the modules and classes chapters of Learning Puppet.

3. In the console, remove all of the previous example classes from your nodes and groups (for example, wsus_client and critical_policy). Be sure to leave the pe_* classes in place.
4. Add the site::basic class to the console with Add new class.
5. Assign the `site::basic` class to the `windows_example` group. Your nodes are now receiving the same configurations as before, but with a simplified interface in the console. Instead of deciding which classes a new node should receive, you can decide what type of node it is and take advantage of decisions you made earlier.

**Summary**

Writing modules enables you to manage your PE configurations.

In this section, you have performed the core workflows of an intermediate Puppet user.

In the course of their normal work, intermediate users:

- Download and modify Forge modules to fit their deployment's needs.
- Create new modules and write new classes to manage many types of resources, including files, services, packages, user accounts, and more.
- Build and curate a site module to safely empower junior admins and simplify the decisions involved in deploying new machines.
- Monitor and troubleshoot events that affect their infrastructure.

**Next Steps**

After you've worked through the Puppet Enterprise (PE) getting started steps, you can now perform the core workflows of a Puppet user.

To continue learning, take the following next steps:

- Read about the different ways you can use Puppet to manage your Windows configurations.
- Learn more about Windows modules and working with the Forge.
- Use the roles and profiles method to create a complete system configuration and store it in a control repository.
- Manage Puppet code using PE Code Manager tools and a control repository.
- Classify nodes using groups and rules.

**Common configuration tasks**

This selection of common configuration tasks gives you just a sample of the things you can manage with Puppet Enterprise. These steps provide an excellent introduction to the capabilities of PE.

PE enables you to manage processes like time syncing, elevated privileges, and firewall rules from one central location.
Managing NTP with PE on page 100
The clocks on your servers need to synchronize with something to let them know what the right time is. NTP is a protocol that synchronizes computer clocks over a network to within a millisecond, using Coordinated Universal Time (UTC). Follow this guide to get time synced across all your PE-managed nodes.

Managing a DNS nameserver with PE on page 103
A nameserver ensures that the human-readable names you type in your browser (for example, google.com) can be resolved to IP addresses that computers can read. This guide provides instructions for getting started managing a simple DNS nameserver file with PE.

Managing SSH with Puppet on page 107
This guide provides instructions for getting started managing SSH across your PE deployment using a module from the Puppet Forge.

Managing sudo with PE on page 111
Managing sudo on your agent nodes allows you to control which system users have access to elevated privileges. This guide provides instructions for getting started managing sudo privileges across your nodes, using a module from the Puppet Forge in conjunction with a simple module you will write.

Managing firewalls with PE on page 113
Follow the steps in this guide to get started managing firewall rules with the puppet-firewall module and a simple module you’ll write that defines those rules.

Managing Windows configurations on page 119
This page covers the different ways you can use Puppet Enterprise (PE) to manage your Windows configurations, including creating local group and user accounts.

Installing and using Windows modules on page 129
This guide covers creating a managed permission with ACL, creating managed registry keys and values with registry, and installing and creating your own packages with chocolatey.

Managing NTP with PE
The clocks on your servers need to synchronize with something to let them know what the right time is. NTP is a protocol that synchronizes computer clocks over a network to within a millisecond, using Coordinated Universal Time (UTC). Follow this guide to get time synced across all your PE-managed nodes.

Your entire datacenter, from the network to the applications, depends on accurate time for many different things, such as security services, certificate validation, and file sharing across nodes.

NTP is one of the most crucial, yet easy, services to configure and manage with Puppet Enterprise. Using the Puppet ntp module, you can:

- Ensure time is correctly synced across all the servers in your infrastructure.
- Ensure time is correctly synced across your configuration management tools.
- Roll out updates quickly if you need to change or specify your own internal NTP server pool.

The ntp module is supported, tested, and maintained by Puppet. You can learn more about the module by visiting the related topic about the ntp module on the Puppet Forge.

The ntp module contains several classes. Classes are named chunks of Puppet code and are the primary means by which Puppet Enterprise configures nodes. The ntp module contains several classes, but the only class that you will use is the ntp class. This class includes several other private classes that are for the module's internal use only.

Related topics:
- Puppet Supported modules
- The ntp module on the Puppet Forge
- About classes in Puppet

NTP overview
To get started managing NTP across your infrastructure, you'll install the Puppet ntp module and then manage it in the console.

To manage your NTP service, you'll do the following tasks:
• Install the `ntp` module.
• Create an NTP node group
• Add the `ntp` class from the module to your agent nodes in the PE console.
• View changes to your infrastructure in the PE console Events page.
• Edit parameters of the main NTP class.

These instructions assume you have installed PE. Refer to the installation overview and the agent installation instructions for complete instructions. See the supported operating system documentation for supported platforms. This guide assumes you are *not* using Code Manager or r10k.

**About module directories**

By default, Puppet keeps modules in `/etc/puppetlabs/code/environments/production/modules`. This includes modules that you download from the Forge and those you write yourself.

PE also creates two other module directories: `/opt/puppetlabs/puppet/modules` and `/etc/puppetlabs/staging-code/modules`. For this guide, don't modify or add anything to either of these directories.

There are plenty of resources about modules and the creation of modules that you can reference.

Related topics:
• Puppet: Module fundamentals.
• Puppet: The modulepath.
• The Beginner's guide to modules.
• The Puppet Forge.

**Install the NTP module**

Install the puppetlabs-ntp module, which helps manage your NTP service.

From the command line of your Puppet master, run `puppet module install puppetlabs-ntp`

You should see output similar to the following:

```
Preparing to install into /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from http://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
### puppetlabs-ntp (v3.1.2)
```

That's it! You've just installed the `ntp` module. You must wait a short time for the Puppet server to refresh before the classes are available to add to your agent nodes.

**Add the `ntp` class from the module**

Using the PE console, you add the module's `ntp` class to a node group that you create, called NTP, which will contain all of your nodes. Depending on your needs or infrastructure, you might have a different group that you'll assign NTP to, but these same instructions would apply.

**Create the NTP node group**

The role of a classification node group is to assign classification data, such as classes, parameters, and variables, to nodes.

1. In the console, click **Classification**, and click **Add group**.
2. Specify options for the new node group:
   • **Parent name**: select All nodes
   • **Group name**: enter a name that describes the role of this environment node group
   • **Environment**: select **production**
   • **Environment group**: don't select this option

3. Click **Add**

4. Click the **NTP** group, and select the **Rules** tab.

5. In the **Fact** field, enter **name**.

6. From the **Operator** drop-down list, select ~ (matches regex).

7. In the **Value** field, enter **.***.

8. Click **Add rule**.

This rule "dynamically" pins all nodes to the **NTP** group. Note that this rule is for testing purposes and that decisions about pinning nodes to groups in a production environment will vary from user to user.

### Add the **ntp** class to the NTP group

Node groups contain classes and other elements.

1. In the console, click **Classification**, and find and select the **NTP** group.

2. On the **Configuration** tab, in the **Add new class** field, select **ntp**.

   **Tip**: You only need to add the main **ntp** class; it contains the other classes from the module.

3. Click **Add class**, and commit changes.

   **Note**: The **ntp** class now appears in the list of classes for the **NTP** group, but it has not yet been configured on your nodes. For that to happen, you need to kick off a Puppet run.

4. From the command line of your Puppet master, run `puppet agent -t`.

5. From the command line of each PE-managed node, run `puppet agent -t`.

   This configures the nodes using the newly-assigned classes.

**Success!** Puppet Enterprise is now managing NTP on the nodes in the **NTP** group. So, for example, if you forget to restart the NTP service on one of those nodes after running `ntpd`, PE will automatically restart it on the next Puppet run.

### View **ntp** changes in the PE console

You can view and research infrastructure changes and events on the console's Events page. After applying the **ntp** class, check the Events page to confirm that changes were indeed made to your infrastructure.

Note that in the summary pane on the left, one event, a successful change, has been recorded for **Nodes**: with events. However, there are two changes for **Classes**: with events and **Resources**: with events. This is because the **ntp** class loaded from the **ntp** module contains additional classes---a class that handles the configuration of NTP (**Ntp::Config**) and a class that handles the NTP service (**Ntp::Service**).

1. Click **Intentional changes in the Classes: with events** summary view. The main pane shows you that the **Ntp::Config** and **Ntp::Service** classes were successfully added when you ran PE after adding the main **ntp** class.

2. Navigate through further levels to see more data.

If you continue to navigate down, you will end up at a run summary that shows you the details of the event. For example, you can see exactly which piece of Puppet code was responsible for generating the event. In this case, it was line 15 of the **service.pp** manifest and line 21 of the **config.pp** manifest from the `puppetlabs-ntp` module.

If there had been a problem applying this class, this information would tell you exactly which piece of code you need to fix. In this case, the **Events** page lets you confirm that PE is now managing NTP.
In the upper right corner of the detail pane is a link to a run report, which contains information about the Puppet run that made the change, including logs and metrics about the run. See Infrastructure reports for more information.

For more information about using the Events page, see Working with the Events page.

**Edit parameters of the ntp class**
You can edit or add class parameters in the PE console without needing to edit the module code directly.

The NTP module, by default, uses public NTP servers. But what if your infrastructure runs an internal pool of NTP servers? You can change the server parameter of the ntp class in a few steps using the PE console.

1. In the console, click Classification, and find and select the NTP group.
2. On the Configuration tab, find ntp in the list of classes.
3. From the Parameter name drop-down list, choose servers.
   
   **Note:** The gray text that appears as values for some parameters is the default value, which can be either a literal value or a Puppet variable. You can restore this value by selecting Discard changes after you have added the parameter.
4. In the Value field, enter the new server name (for example, ["time.apple.com"]). Note that this should be an array, in JSON format.
5. Click Add parameter, and commit changes.
6. From the command line of your Puppet master, run puppet agent -t.
7. From the command line of each PE-managed node, run puppet agent -t.

This triggers a Puppet run that causes Puppet Enterprise to create the new configuration.

Puppet Enterprise will now use the NTP server you've specified for that node.

**Tip:** Remember to check the Events page to be sure the changes were correctly applied to your nodes!

**Learning more about Puppet and the ntp module**
You can learn more about Puppet and Puppet supported modules.

For more information about working with the Puppet ntp module, check out our puppetlabs-ntp: A Puppet Enterprise supported module blog post.

Puppet offers many opportunities for learning and training, from formal certification courses to guided online lessons. Head over to the Learning page to discover more.

**Managing a DNS nameserver with PE**
A nameserver ensures that the human-readable names you type in your browser (for example, google.com) can be resolved to IP addresses that computers can read. This guide provides instructions for getting started managing a simple DNS nameserver file with PE.

Sysadmins typically need to manage a nameserver file for internal resources that aren't published in public nameservers. For example, let's say you have several employee-maintained servers in your infrastructure, and the DNS network assigned to those servers use Google's public nameserver located at 8.8.8.8. However, there are several resources behind your company's firewall that your employees need to access on a regular basis. In this case, you'd build a private nameserver (say at 10.16.22.10), and then use PE to ensure all the servers in your infrastructure have access to it.

**DNS getting started overview**
To get started managing the DNS nameserver, you'll create a Puppet module and then manage it in the console.

In this guide, you'll do the following tasks:

- Write a simple module that contains a class called resolver to manage a nameserver file called, /etc/resolv.conf.
- Create a DNS node group.
- Add the resolver class to your agent nodes in the PE console.
• Change the contents of the nameserver file to see how PE enforces the desired state you specified in the PE console.

Before you begin, you must have installed PE. Refer to the installation overview and the agent installation instructions for complete instructions. See the supported operating system documentation for supported platforms. This guide assumes you are not using Code Manager or r10k.

Tip: Follow the instructions in the NTP getting started guide to have PE ensure time is in sync across your deployment.

Note: You can add the DNS nameserver class to as many agents as needed. For ease of explanation, our console images and instructions might show only one agent.

About module directories
By default, Puppet keeps modules in /etc/puppetlabs/code/environments/production/modules. This includes modules that you download from the Forge and those you write yourself.

PE also creates two other module directories: /opt/puppetlabs/puppet/modules and /etc/puppetlabs/staging-code/modules. For this guide, don't modify or add anything to either of these directories.

There are plenty of resources about modules and the creation of modules that you can reference.

Related topics:
• Puppet: Module fundamentals.
• Puppet: The modulepath.
• The Beginner's guide to modules.
• The Puppet Forge.

Write the resolver module
Write a small module to ensure that your nodes resolve to your internal nameserver.

This module contains just one class and one template. Modules are directory trees. For this task, you'll create the following files:

• resolver/ (the module name)
  • manifests/
    • init.pp (contains the resolver class)
  • templates/
    • resolve.conf.erb (contains the template for the /etc/resolv.conf template, the contents of which will be populated after you add the class and run PE.)

1. From the command line on the Puppet master, navigate to the modules directory: cd /etc/puppetlabs/code/environments/production/modules
2. Run mkdir -p resolver/manifeest
3. From the manifests directory, use your text editor to create the init.pp file, and edit it so it contains the following Puppet code.

```puppet
class resolver ($nameservers,)
{

    file { '/etc/resolv.conf':
        ensure => file,
        owner  => 'root',
        group  => 'root',
        mode   => '0644',
        content => template('resolver/resolv.conf.erb'),
    }
}
```

4. Save and exit the file.
5. Run `mkdir -p resolver/templates` to create the templates directory.
6. Use your text editor to create the `resolver/templates/resolv.conf.erb` file.
7. Edit the `resolv.conf.erb` so that it contains the following Ruby code.

```ruby
# Resolv.conf generated by Puppet
<% @nameservers.flatten.each do |ns| -%>
nameserver <%= ns %>
<% end -%>

# Other values can be added or hard-coded into the template as needed.
```

8. Save and exit the file.

That's it! You've written a module that contains a class that will, once applied, ensure your nodes resolve to your internal nameserver. You'll need to wait a short time for the Puppet server to refresh before the classes are available to add to your agents.

Note the following about your new class:

- The class `resolver` ensures the creation of the file `/etc/resolv.conf`.
- The content of `/etc/resolv.conf` is modified and managed by the template, `resolv.conf.erb`. You will set this content in the next task using the PE console.

**Create the DNS node group**

To manage DNS on your nodes, create a new node group that contains all of your nodes.

Create the DNS node group to contain all the nodes in your deployment (including the Puppet master). You can create your own groups or add the classes to individual nodes, depending on your needs.

1. In the console, click **Classification**, and click **Add group**.
2. Specify options for the new node group:
   - **Parent name** – Select default
   - **Group name** - Enter a name that describes the role of this environment node group, for example, DNS.
   - **Environment** - Select Production
   - **Environment group** - Don't select this option
3. Click **Add**.
4. Click the DNS group, and select the **Rules** tab.
5. In the **Fact** field, enter `name`.
6. From the **Operator** drop-down list, select `~` (matches regex).
7. In the **Value** field, enter `.*`. 

© 2020 Puppet, Inc.
8. Click **Add rule**.

This rule "dynamically" pins all nodes to the DNS group. This rule is for testing purposes; decisions about pinning nodes to groups in a production environment will vary from user to user.

**Add the resolver class to the DNS group**

After you create a group, add a class to it.

Next, you'll add the resolver class to your new DNS node group.

1. In the console, select **Classification**, and then find and select the **DNS** group.
2. On the **Configuration** tab, in the **Class name** field, select **resolver**.
3. Click **Add class**, and commit changes.

**Note:** The resolver class now appears in the list of classes for the DNS group, but it has not yet been configured on your nodes. For that to happen, you need to kick off a Puppet run.

4. From the command line of your Puppet master, run `puppet agent -t`.
5. From the command line of each PE-managed node, run `puppet agent -t`.

This will configure the nodes using the newly-assigned classes. Wait one or two minutes.

You're not done just yet! The resolver class now appears in the list of classes for your DNS group, but it has not yet been fully configured. You still need to add the nameserver IP address parameter for the resolver class to use. You can do this by adding a parameter right in the console.

**Add the nameserver IP address parameter in the console**

You can add class parameter values to the code in your module, but it’s easier to add those parameter values to your classes using the PE console.

1. In the console, select **Classification**, and then find and select the **DNS** group.
2. On the **Configuration** tab, find resolver in the list of classes.
3. From the **parameter** drop-down list, select **nameservers**.
4. In the **Value** field, enter the nameserver IP address you’d like to use (for example, 8.8.8.8).

**Note:** The gray text that appears as values for some parameters is the default value, which can be either a literal value or a Puppet variable. You can restore this value by selecting **Discard changes** after you have added the parameter.

5. Click **Add parameter**, and commit changes.
6. From the command line of your Puppet master, run `puppet agent -t`.
7. From the command line of each PE-managed node, run `puppet agent -t`.

This triggers a Puppet run to have Puppet Enterprise create the new configuration.

8. Navigate to `/etc/resolv.conf`. This file now contains the contents of the `resolv.conf.erb` template and the nameserver IP address you added in step 5.

Success! Puppet Enterprise will now use the nameserver IP address you've specified for that node.

**Viewing DNS changes on the Events page**

The Events page lets you view and research changes. You can view changes by class, resource, or node.

After applying the resolver class, you can use the Events page to confirm that changes were indeed made to your infrastructure, most notably that the class created `/etc/resolv.conf` and set the contents as specified by the module's template.

The further you drill down in this page, the more detail you'll receive. If there had been a problem applying the resolver class, this information would tell you exactly where that problem occurred or which piece of code you need to fix.

You can click **Reports**, which contains information about the changes made during Puppet runs, including logs and metrics about the run. See Infrastructure reports for more info.

© 2020 Puppet, Inc.
For more information about using the Events page, see Working with the Events page.

**Check that PE enforces the desired state of the resolver class**

If your infrastructure changes from what you've specified, PE will correct that change. To test this, make a manual infrastructure change and then run Puppet.

When you set up DNS nameserver management, you set the nameserver IP address. If a member of your team changes the contents of `/etc/resolv.conf` to use a different nameserver, blocking access to internal resources, the next Puppet run corrects this. You can test this by manually changing the `resolv.conf` file.

1. On any agent to which you applied the `resolv.conf`, edit `/etc/resolv.conf` to be any nameserver IP address other than the one you want to use.
2. Save and exit the file.
3. After Puppet runs, navigate to
4. Puppet runs.
5. Navigate to `/etc/resolv.conf`, and notice that PE has enforced the desired state you specified for the nameserver IP address.

That's it! PE has enforced the desired state of your agent node. And remember, review the changes to the class or node using the Events page.

**Learning more about Puppet and DNS**

This guide provides a basic discussion of Puppet and DNS.

For more information about working with Puppet Enterprise and DNS, check out our Dealing with name resolution issues blog post.

**Managing SSH with Puppet**

This guide provides instructions for getting started managing SSH across your PE deployment using a module from the Puppet Forge.

**About SSH**

Secure Shell (SSH) is a protocol that enables encrypted connections between nodes on a network for administrative purposes.

SSH is most commonly used on *nix systems by admins who wish to remotely log into machines to access the command line and execute commands and scripts.

Typically, the first time you attempt to SSH into a host you’ve never connected to before, you get a warning similar to the following:

```
The authenticity of host '10.10.10.9 (10.10.10.9)' can't be established.
Are you sure you want to continue connecting (yes/no)?
```

If you select yes, the public key for that host is added to your SSH `known_hosts` file, and you won't have to authenticate it again unless that host's key changes.

**Managing SSH overview**

Install an SSH module that uses Puppet resources to collect and distribute the public key for each agent node in your Puppet Enterprise deployment. This enables you to SSH to and from any node without authentication warnings.

You install the `ghoneycutt-ssh` module, a simple module with one class that Puppet Enterprises uses to configure your nodes: the `ssh` class. Classes are named chunks of Puppet code and are the primary means by which Puppet Enterprise configures nodes.

The `ghoneycutt-ssh` module, available on the Puppet Forge, is one of many modules written by our user community. You can learn more about this module by visiting the Forge.

© 2020 Puppet, Inc.
Then, in the PE console, you create a group of the nodes you want to manage SSH on. Groups let you assign variables and classes, such as the `ssh` class, to many nodes at once. Nodes can belong to many groups and inherit classes and variables from all of them. Groups can also be members of other groups and inherit configuration information from their parent group the same way nodes do. PE creates several groups automatically, which you can read more about in the related topic about groups in the console.

Using this guide, you:

- Install the `ghoneycutt-ssh` module.
- Create the SSH node group
- Add classes from the `ssh` module to your agent nodes in the console.
- View changes to your infrastructure in the console **Events** page.
- Edit root login parameters of the `ssh` class in the console.

These instructions assume you have installed PE. Refer to the installation overview and the agent installation instructions for complete instructions. See the supported operating system documentation for supported platforms. This guide assumes you are *not* using Code Manager or r10k.

Related topics:

- Groups in the console
- `ghoneycutt-ssh` on the Forge
- About classes

**About module directories**

By default, Puppet keeps modules in `/etc/puppetlabs/code/environments/production/modules`. This includes modules that you download from the Forge and those you write yourself.

PE also creates two other module directories: `/opt/puppetlabs/puppet/modules` and `/etc/puppetlabs/staging-code/modules`. For this guide, don't modify or add anything to either of these directories.

There are plenty of resources about modules and the creation of modules that you can reference.

Related topics:

- Puppet: Module fundamentals.
- Puppet: The modulepath.
- The Beginner's guide to modules.
- The Puppet Forge.

**Install the ghoneycutt-ssh module**

The `ghoneycutt-ssh` module manages SSH keys and removes SSH keys that you aren't managing with Puppet.

This module, available on the Puppet Forge, is one of many modules written by our user community. It contains one class: the `ssh` class. You can learn more about the `ghoneycutt-ssh` module by visiting the Forge.

On the command line, on the PE master, run `puppet module install ghoneycutt-ssh --version 3.40.0`.

You should see output similar to the following:

```
Notice: Preparing to install into /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
### ghoneycutt-ssh (v3.40.0)
### ghoneycutt-common (v1.6.0)
### puppetlabs-firewall (v1.8.1)
### puppetlabs-stdlib (v4.9.1)
```
That's it! You've just installed the ghoneycutt-ssh module. You'll need to wait a short time for the Puppet server to refresh before the classes are available to add to your agent nodes.

Create the SSH node group
In the console, create a group called ssh_example to designate which nodes Puppet should manage SSH on.

This group will contain all of your nodes. Depending on your needs or infrastructure, you might have a different group that you'll assign SSH to.

1. In the console, click Classification, and click Add group.
2. Specify options for the new node group:
   - Parent name: select All nodes
   - Group name: enter a name that describes the role of this environment node group
   - Environment: select production
   - Environment group: don't select this option
3. Click Add
4. Click the ssh_example group, then select the Rules tab.
5. In the Fact field, enter name.
6. From the Operator drop-down list, select ~ (matches regex).
7. In the Value field, enter .*.
8. Click Add rule.

This rule "dynamically" pins all nodes to the ssh_example group. Note that this rule is for testing purposes, and that decisions about pinning nodes to groups in a production environment vary.

Related topics:
Adding nodes dynamically

Add classes from the ssh module
Add the ssh class to the ssh_example node group to add the necessary resources that allow Puppet to manage SSH.

Depending on your needs or infrastructure, you may have a different group that you'll assign SSH to, but these same instructions would apply.

After you apply the ssh class and run Puppet, the public key for each agent node will be exported and then disseminated to the known_hosts files of the other agent nodes in the group, and you will no longer be asked to authenticate those nodes on future SSH attempts.

1. In the console, click Classification, and find and select ssh_example group.
2. On the Configuration tab, in the Class name field, select ssh.
3. Click Add class, and commit changes.

   Note: The ssh class now appears in the list of classes for the ssh_example group, but it has not yet been configured on your nodes. For that to happen, kick off a Puppet run.

4. From the command line of your master, run puppet agent -t.
5. From the command line of each -managed node, run puppet agent -t.

This configures the nodes using the newly assigned classes. Wait one or two minutes.

Important: You need to run Puppet a second time due to the round-robin nature of the key sharing. In other words, the first server that ran on the first Puppet run was only able to share its key, but it was not able to retrieve the keys from the other agents. It will collect the other keys on the second Puppet run.
View changes made by the ssh class
To confirm that the ssh class made changes to your infrastructure, check the Events console page. This page lets you view and research changes and other events.

For example, after applying the ssh class, you can use the Events page to confirm that changes were indeed made to your infrastructure.

Note that in the summary pane on the left, one event, a successful change, has been recorded for Classes: with events. However, there are three changes for Classes: with events and six changes for Resources: with events.

1. Click With changes in the Classes: with events summary view.
   The main pane will show you that the ssh class was successfully added when you ran PE. This class sets the known_hosts entries after it collects the public keys from agents nodes in your deployment.

2. Click Changed in the Resources: with events summary view.
   The main page shows you that public key resources for each agent in our example has now been brought under PE management. The further down you navigate, the more information you receive about the event. For example, in this case, you see that the the SSH rsa key for agent1.example.com has been created and is now present in the known_hosts file for master.example.com.

If there had been a problem applying any piece of the ssh class, the information found here could tell you exactly which piece of code you need to fix. In this case, the Events page simply lets you confirm that PE is now managing SSH keys.

In the upper right corner of the detail pane is a link to a run report which contains information about the Puppet run that made the change, including logs and metrics about the run. See Infrastructure reports for more information.

For more information about using the Events page, see Working with the Events page.

Edit root login parameters of the ssh class
Edit or add class parameters from the ssh class in the PE console without needing to edit the module code directly.

The ghoneycutt-ssh module, by default, allows root login over SSH. But if your compliance protocols do not allow this, you can change this parameter of the ssh class in a few steps using the PE console.

1. In the console, click Classification, and find and select the ssh_example group.
2. On the Configuration tab, find ssh in the list of classes.
3. From the parameter drop-down menu, choose permit_root_login.
   Note: The gray text that appears as values for some parameters is the default value, which can be either a literal value or a Puppet variable. You can restore this value by clicking Discard changes after you have added the parameter.
4. In the Value field, enter no.
5. Click Add parameter, and commit changes.
6. From the command line of your master, run puppet agent -t.
7. From the command line of each -managed node, run puppet agent -t.

Puppet Enterprise is now managing the root login parameter for your SSH configuration. You can see this setting in /etc/ssh/sshd_config. For fun, change the PermitRootLogin parameter to yes, run PE, and then recheck this file. As long as the parameter is set to no in the PE console, the parameter in this file will be set back to no on every Puppet run if it is ever changed.

You can use the PE console to manage other SSH parameters, such as agent forwarding, X11 forwarding, and password authentication.

Other SSH resources
Find blog posts and other SSH information on Puppet.com.

For a video on automating SSH with Puppet Enterprise, check out Automate SSH configuration in 5 minutes with Puppet Enterprise.

© 2020 Puppet, Inc.
Speed up SSH by reusing connections on the Puppet blog gives some helpful hints for working with SSH.

**Managing sudo with PE**

Managing sudo on your agent nodes allows you to control which system users have access to elevated privileges. This guide provides instructions for getting started managing sudo privileges across your nodes, using a module from the Puppet Forge in conjunction with a simple module you will write.

In most cases, you want to manage sudo on your nodes to control which system users have access to elevated privileges.

**Managing sudo overview**

To manage sudo configuration and privileges across your deployment, you write a module.

The *saz-sudo* module, available on the Puppet Forge, is one of many modules written by a member of our user community. To learn more about the module, visit the Puppet Forge.

You also write a simple *privileges* module with a class to manage sudo privileges. The *saz-sudo* module has several classes, but the module you write contains just one.

- Classes
- *saz-sudo* on Puppet Forge.

Using this guide, you:

- Install the *saz-sudo* module as the foundation for your management of sudo privileges.
- Write a simple *privileges* module to manage a few resources that set privileges for certain users, which will be managed by the *saz-sudo* module.
- Create a *Sudo* node group
- Add classes from the *privileges* and *sudo* modules to your agent nodes in the console.

These instructions assume you have installed PE. Refer to the installation overview and the agent installation instructions for complete instructions. See the supported operating system documentation for supported platforms.

This guide assumes that you are *not* using Code Manager or r10k.

**Note:** Follow the instructions in the NTP getting started guide to have PE ensure time is in sync across your deployment.

**About module directories**

By default, Puppet keeps modules in `/etc/puppetlabs/code/environments/production/modules`. This includes modules that you download from the Forge and those you write yourself.

PE also creates two other module directories: `/opt/puppetlabs/puppet/modules` and `/etc/puppetlabs/staging-code/modules`. For this guide, don't modify or add anything to either of these directories.

There are plenty of resources about modules and the creation of modules that you can reference.

**Install the saz-sudo module**

To start managing sudo configuration with Puppet Enterprise, install the *saz-sudo* module.

From the master, run `puppet module install saz-sudo`. 

© 2020 Puppet, Inc.
You should see output similar to the following:

Preparing to install into /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from http://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ... /etc/puppetlabs/code/environments/production/modules ### saz-sudo (v2.3.6) ### puppetlabs-stdlib (3.2.2) [/opt/puppetlabs/puppet/modules]

That's it! You've just installed the saz-sudo module. Wait a short time for the Puppet server to refresh before the classes are available to add to your agents.

Write the privileges module
Manage sudo privileges with Puppet Enterprise, by writing a privileges module.

The privileges module will contain the following files:

- privileges/ (the module name)
- manifests/
  - init.pp (contains the privileges class)

1. From the command line on the master, navigate to the modules directory:
   cd /etc/puppetlabs/code/environments/production/modules
2. Run mkdir -p privileges/manifests to create the new module directory and its manifests directory.
3. From the manifests directory, use your text editor to create the init.pp file, and edit it so that it contains the following code.

   ```puppet
class privileges { 
   user { 'root': ensure => present, password => '$1$oST1TkX7$p21hU2qzMkR4Iy7HK6zWq0', shell => '/bin/bash', uid => '0', } 
   sudo::conf { 'admins': ensure => present, content => '%admin ALL=(ALL) ALL', } 
   sudo::conf { 'wheel': ensure => present, content => '%wheel ALL=(ALL) ALL', } 
   }
```

4. Save and exit the file.

That's it! You've written a module that contains a class that, when applied, ensures that your agent nodes have the correct sudo privileges set for the root user and the "admin" and "wheel" groups. You will add this class at the same time you add the saz-sudo module.

To learn more about what the resources in this class do, see the related topic about the resources in the privileges class.

About the resources in the privileges class
The privileges module you wrote for managing sudo privileges in your deployment contains just one class, but several resources. Each resource has a specific job.

The privileges module contains the following resources:

- user 'root': This resource ensures that the root user has a centrally defined password and shell. Puppet enforces this configuration and report on and remediate any drift detected, such as if a rogue admin logs in and changes the password on an agent node.
- sudo::conf 'admins': Create a sudoers rule to ensure that members of the admin group have the ability to run any command using sudo. This resource creates configuration fragment file to define this rule in /etc/sudoers.d/. It is usually called something like 10_admins.
- sudo::conf 'wheel': Create a sudoers rule to ensure that members of the wheel group have the ability to run any command using sudo. This resource creates a configuration fragment to define this rule in /etc/sudoers.d/. It is usually called something like 10_wheel.

© 2020 Puppet, Inc.
Create the Sudo node group
To specify which nodes you want to manage sudo on, set up a designated node group.

This group, called Sudo, will contain all of your nodes. Depending on your needs or infrastructure, your group might be different.

1. In the console, click Classification, and click Add group.
2. Specify options for the new node group:
   - **Parent name**: select All nodes
   - **Group name**: enter a name that describes the role of this environment node group
   - **Environment**: select production
   - **Environment group**: don't select this option
3. Click Add
4. Click the Sudo group and select the Rules tab.
5. In the Fact field, enter name.
6. From the Operator drop-down list, select ~ (matches regex).
7. In the Value field, enter .*.
8. Click Add rule.

This rule "dynamically" pins all nodes to the Sudo group. Note that this rule is for testing purposes and that decisions about pinning nodes to groups in a production environment vary. To learn more, see the related topic about dynamically pinning nodes.

Related topics: Adding nodes "dynamically"

Add the privileges and sudo classes
To manage sudo configuration and privileges for the nodes in your Sudo group, add the privileges and sudo classes to your node group.

The privileges module you wrote has only one class (privileges), but the saz-sudo module contains several classes. If you don't want to add these classes to all of your nodes, you can pin the nodes "statically" or write a different rule to add them "dynamically", depending on your needs. See the related topics about adding nodes dynamically or statically for more information.

1. In the console, click Classification, and find and select the Sudo group.
2. On the Configuration tab, in the Class name field, enter sudo.
3. Click Add class, and commit changes.
   - **Note**: The sudo class now appears in the list of classes for the Sudo group, but it has not yet been configured on your nodes. For that to happen, you need to kick off a Puppet run.
4. Repeat steps 2 and 3 to add the privileges class.
5. From the command line of your master, run puppet agent -t.
6. From the command line of each -managed node, run puppet agent -t.

This configures the nodes using the newly-assigned classes. Wait one or two minutes.

Congratulations! You’ve just created the privileges class that you can use to define and enforce a sudoers configuration across your PE-managed infrastructure.

Managing firewalls with PE
Follow the steps in this guide to get started managing firewall rules with the puppet-firewall module and a simple module you’ll write that defines those rules.
Managing firewalls overview
Firewalls consist of a set of rules and policies for managing access.

With a firewall, admins define a set of policies (firewall rules) that usually consist of things like application ports (TCP/UDP), node interfaces (which network port), IP addresses, and an accept/deny statement. These rules are applied from a "top-to-bottom" approach.

For example, when a service, such as SSH, attempts to access resources on the other side of a firewall, the firewall applies a list of rules to determine if or how SSH communications are handled. If a rule allowing SSH access can’t be found, the firewall denies access to that SSH attempt.

To best manage firewall rules with PE, separate these rules into pre and post groups.

Learning to manage firewalls with PE
To manage your firewall rules with Puppet Enterprise, define a group of nodes that you want to manage, and then use modules to manage the rules on those nodes.

Groups let you assign classes and variables to many nodes at once. Nodes can belong to many groups and inherit classes and variables from all of them. Groups can also be members of other groups and inherit configuration information from their parent group the same way nodes do. PE automatically creates several groups in the console.

To set up firewall management with Puppet Enterprise, you will:

• Write a simple module to define the firewall rules for your PE-managed infrastructure.
• Create a firewall node group
• Add the my_firewall class to your agent nodes.
• Write an additional class to open ports for the Puppet master
• Enforce the desired state of the my_firewall class.

These instructions assume that you have installed PE. Refer to the installation overview and the agent installation instructions for complete instructions. See the supported operating system documentation for supported platforms. This guide assumes you are not using Code Manager or r10k.

Tip: Follow the instructions in the NTP getting started guide to have PE ensure time is in sync across your deployment.

Related topics:
• Preconfigured groups in the console.

About module directories
By default, Puppet keeps modules in /etc/puppetlabs/code/environments/production/modules. This includes modules that you download from the Forge and those you write yourself.

PE also creates two other module directories:/opt/puppetlabs/puppet/modules and /etc/puppetlabs/staging-code/modules. For this guide, don’t modify or add anything to either of these directories.

There are plenty of resources about modules and the creation of modules that you can reference.

Related topics:
• Puppet: Module fundamentals.
• Puppet: The modulepath.
• The Beginner's guide to modules.
• The Puppet Forge.

Install the puppetlabs-firewall module
Install the puppetlabs-firewall module to manage firewall policies.

To get started managing and configuring firewall rules with Puppet, install the puppetlabs-firewall module.

From the PE master, run puppet module install puppetlabs-firewall
You should see output similar to the following:

Preparing to install into /etc/puppetlabs/puppet/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/puppet/environments/production/modules
### puppetlabs-firewall (v1.6.0)

That's it! You've just installed the firewall module. You'll need to wait a short time for the Puppet server to refresh before the classes are available to add to your agent nodes.

Related topics: Firewall

**Write the my_firewall module**

Create a simple firewall module.

To define and help manage your firewall rules, you'll write a simple module containing just three classes.

For your module, you'll create the following files:

- my_firewall/
  - manifests/
    - pre.pp
    - post.pp
    - init.pp

1. From the command line on the Puppet master, navigate to the modules directory:
   ```
   cd /etc/puppetlabs/code/environments/production/modules
   ```
2. Run `mkdir -p my_fw/manifests` to create the new module directory and its manifests directory.
3. From the manifests directory, use your text editor to create `pre.pp`. 
4. Edit `pre.pp` so it contains the following Puppet code.

    class my_firewall::pre {
        # Default firewall rules
        firewall { '000 accept all icmp':
            proto   => 'icmp',
            action  => 'accept',
        }

        firewall { '001 accept all to lo interface':
            proto   => 'all',
            iniface => 'lo',
            action  => 'accept',
        }

        firewall { '002 accept related established rules':
            proto   => 'all',
            state   => ['RELATED', 'ESTABLISHED'],
            action  => 'accept',
        }

        # Allow SSH
        firewall { '100 allow ssh access':
            port   => '22',
            proto  => tcp,
            action => accept,
        }
    }

The `pre.pp` file defines the "pre" group rules that the firewall applies when a service requests access.

5. Save and exit the file.

6. From the `manifests` directory, use your text editor to create `post.pp`.

7. Edit `post.pp` so that it contains the following Puppet code.

    class my_firewall::post {
        firewall { "999 drop all other requests":
            action => "drop",
        }
    }

The `post.pp` file defines the rule for the firewall to drop any requests that haven’t met the rules defined by `pre.pp`.

8. Save and exit the file.

9. From the `manifests` directory, use your text editor to create `init.pp`. 
10. Edit the init.pp file so that it contains the following Puppet code.

```puppet
class my_firewall {
  stage { 'fw_pre': before => Stage['main']; }
  stage { 'fw_post': require => Stage['main']; }

  class { 'my_fw::pre':
    stage => 'fw_pre',
  }

  class { 'my_fw::post':
    stage => 'fw_post',
  }

  resources { "firewall":
    purge => true
  }
}
```

The init.pp file applies the previous two classes, as well as telling Puppet when to apply the classes in relation to the main stage (which ensures the classes are applied in the correct order).

11. Save and exit the file.

That's it! You've written a module that contains a class that, once applied, ensures your firewall has rules that will be managed by PE. You'll need to wait a short time for the Puppet server to refresh before the classes are available to add to your agent nodes.

Create the firewall_example group

Create a group of nodes that you want to manage firewalls for.

To specify the nodes you want to manage firewalls on, create a group called firewall_example. This group contains all of your nodes. Depending on your needs or infrastructure, you may have a different group that you assign your firewall class to.

1. In the console, click Classification, and click Add group.

2. In the Group name field, name your group, for example firewall_example, and click Add.

3. Click Add membership rules, classes, and variables.

4. On the Rules tab, in the Node name field, enter the name of the PE-managed node you want to add to this group, and click Pin node.

   Repeat this step for any additional nodes you want to add.

   **CAUTION:** Do not add the Puppet master to this group. Your firewall class does not yet contain rules to allow access to the Puppet master. You will add these rules later when you write a class to open ports for the Puppet master.

5. Commit changes.

Add the my_firewall class to agent nodes

Enforce firewall rules by adding the firewall class to the nodes you want to manage.

To define and enforce firewall rules across your agent notes, add the my_firewall class from your module to the node group.

1. In the console, click Classification, and find and select the firewall_example group.

2. On the Configuration tab, in the Class name field, select my_firewall.

   **Tip:** You only need to add the main my_firewall class. It contains the other classes from the module.
3. Click **Add class**.

   The *my_firewall* class now appears in the list of classes for the *firewall_example* group.

   When Puppet runs, it configures the nodes using the newly-assigned classes. Wait one or two minutes.

Congratulations! You’ve just created a firewall class that you can use to define and enforce firewall rules across your PE-managed infrastructure.

**Open ports for the Puppet master**

When creating firewall policies, you must allow special access to the Puppet master.

To ensure that you can access the Puppet master correctly, you’ll need to allow special firewall access. To do this, create a module that opens the ports for the Puppet master.

1. From the command line on the master, navigate to the modules directory:
   ```bash
cd /etc/puppetlabs/code/environments/production/modules
   ```
2. Run `mkdir -p privileges/manifests` to create the new module directory and its manifests directory.
3. From the manifests directory, use your text editor to create the `init.pp` file, and edit it so it contains the following Puppet code.

   ```puppet
class my_master {
    include my_firewall

    firewall { '100 allow PE Console access':
        port => '443',
        proto => tcp,
        action => accept,
    }

    firewall { '100 allow Puppet master access':
        port => '8140',
        proto => tcp,
        action => accept,
    }

    firewall { '100 allow ActiveMQ MCollective access':
        port => '61613',
        proto => tcp,
        action => accept,
    }
}
```

4. Using the console, add *my_master* class to the Puppet master.
5. Add the Puppet master to the *firewall_example* group.
6. After Puppet runs, navigate to the **Reports** page, and click the latest run report for your Puppet master.
7. Click the **Events** tab.

   You will see three events on this node, indicating that your firewall now allows access to MCollective, the PE console, and the Puppet master.

   **Tip:** For all firewall configuration needs for your PE installation, refer to the system configuration documentation.

**Check that PE enforces the desired state of the **my_firewall** class**

If your infrastructure changes from what you’ve specified, PE will correct that change.

To test that PE enforces the desired state of the *my_firewall* class you created, make a manual change and then run Puppet.

---

© 2020 Puppet, Inc.
For example, you applied the class `my_firewall` to define and enforce your firewall rules. If a member of your team changes the contents of the `iptables` to allow connections on a random port that is not specified in `my_firewall`, PE corrects the change the next time it runs. You can test this by making a manual change.

1. Select an agent node on which you applied the `my_firewall` class, and run `iptables --list`.
2. Note that the rules from the `my_firewall` class have been applied.
3. From the command line, insert a new rule to allow connections to port 8449 by running `iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport 8449 -j ACCEPT`.
4. Run `iptables --list` again and note this new rule is now listed.
5. After Puppet runs on the agent node, run `iptables --list` on that node once more, and notice that PE has enforced the desired state you specified for the firewall rules.

That's it --- PE has enforced the desired state of your agent node.

**Learning more about Puppet Enterprise and puppetlabs-firewall**

The Puppet firewall module (`puppetlabs-firewall`), is part of the PE supported modules program; these modules are supported, tested, and maintained by Puppet.

You can learn more about the Puppet firewall module by visiting the Puppet Forge.

Check out the other getting started guides in our PE getting started guide series:

- NTP getting started guide
- SSH getting started guide
- DNS getting started guide
- Sudo users getting started guide

Puppet offers many opportunities for learning and training, from formal certification courses to guided online lessons. We've noted a few below; head over to the Learning Puppet page to discover more.

- Learning Puppet is a series of exercises on various core topics about deploying and using PE.
- The Puppet workshop contains a series of self-paced, online lessons that cover a variety of topics on Puppet basics. You can sign up at the learning page.

**Managing Windows configurations**

This page covers the different ways you can use Puppet Enterprise (PE) to manage your Windows configurations, including creating local group and user accounts.

**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 to create manifest files. Atom, Visual Studio Code and Sublime Text support 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:

```ruby
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
   ```

**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 will 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:

```bash
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 \texttt{puppet apply}

When the output of the simulation shows the changes you intend to make, you can start enforcing these changes with the \texttt{puppet apply} command.

Run \texttt{puppet apply c:\myfiles\file.pp}.

To see more details about what this command did, you can specify additional options, such as \texttt{--trace}, \texttt{--debug}, or \texttt{--verbose}, which can help you diagnose problematic code. If \texttt{puppet apply} fails, Puppet outputs a full stack trace.

Puppet enforces the resource state you've described in \texttt{file.pp}, in this case guaranteeing that a file (c:\Temp\foo.txt) is present and has the contents:

\texttt{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 \texttt{c:\myfiles\file.pp} to the following:

\begin{verbatim}
file { 'C:\Temp\foo.txt':
  ensure   => present,
  content  => 'I have changed my file content.'
}
\end{verbatim}

Apply the manifest by running \texttt{puppet apply c:\myfiles\file.pp}. Open \texttt{c:\Temp\foo.txt} and notice that Puppet changes the file's contents.

Applying the manifest again with \texttt{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 \texttt{puppet apply}.

Additional command line tools

Once you understand how to write manifests, validate them, and use \texttt{puppet apply} to enforce your changes, you're ready to use commands such as \texttt{puppet agent}, \texttt{puppet resource}, and \texttt{puppet module install}.

\texttt{puppet agent}

Like \texttt{puppet apply}, the \texttt{puppet agent} command line tool applies configuration changes to a system. However, \texttt{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 master every 30 minutes by running \texttt{puppet agent} to retrieve new catalogs and apply them locally.

\texttt{puppet resource}

You can run \texttt{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 \texttt{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 master 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:

```bash
service { '<service name>':
  ensure => 'running'
}
```
Example
For example, the following manifest code ensures the Windows Time service is running:

```yaml
service { 'w32time':
  ensure => 'running'
}
```

Stop a Windows service
Some services can impair performance, or may need to be stopped for regular maintenance.
To disable a service, use the code:

```yaml
service { '<service name>':
  ensure => 'stopped',
  enable => 'false'
}
```

Example
For example, this disables the disk defragmentation service, which can negatively impact service performance.

```yaml
service { 'defragsvc':
  ensure => 'stopped',
  enable => 'false'
}
```

Schedule a recurring task
Regularly scheduled tasks are often necessary on Windows to perform routine system maintenance.
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:

```yaml
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:
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:

```
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:
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:

```yaml
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:

```puppet
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']
}

exec { '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 will make sure the aspnet_banking_app user is a part of the IIS_IUSRS and Users group, but will not 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:

```ruby
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 task 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 will note perform the command to 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.

```
$drive = 'C:'
exec { 'disable-c-indexing':
  command => template('Disable-Indexing.ps1.erb'),
  provider => powershell,
  unless => "if ((Get-WmiObject -Class Win32_Volume -Filter 'DriveLetter=\"${drive}\"').IndexingEnabled) { exit 1 }",
}
```
The PowerShell code for Disable-Indexing.ps1.erb becomes:

```powershell
function Disable-Indexing($Drive)
{
    $drive = Get-WmiObject -Class Win32_Volume -Filter "DriveLetter='$Letter'"
    if ($drive.IndexingEnabled -ne $True) { return }
    $drive | Set-WmiInstance -Arguments @{IndexingEnabled=$False} | Out-Null
}
Disable-Indexing -Drive '<%= @driveLetter %>'
```

## Installing and using Windows modules

This guide covers creating a managed permission with ACL, creating managed registry keys and values with registry, and installing and creating your own packages with chocolatey.

### Related information

- [Module basics](#) on page 96
- Modules are directory trees. Many modules contain more than one directory.

### Windows module pack

The Windows module pack is a group of modules available on the Forge curated to help you complete common Windows tasks.

The Forge is an online community of Puppet modules submitted by Puppet and community members. The Forge makes it easier for you to manage Puppet and can save you time by using pre-written modules, rather than writing your own. In addition to being rated by the community, modules in the Forge can be Puppet Approved or Puppet Supported. The major difference is that Approved modules are not available for Puppet Enterprise support services, but are still tested and adhere to a standard for style and quality.

The Windows module pack includes several Windows compatible modules that help you complete common specific tasks. You can find more Windows modules by searching the Forge. While the module pack itself is not supported, the modules by Puppet contained in the pack are individually supported with PE. The rest have been reviewed and recommended by Puppet but are not eligible for commercial support.

The Windows module pack enables you to do the following:

- Read, create, and write registry keys with registry.
- Interact with PowerShell through the Puppet DSL with powershell.
- Manage Windows PowerShell DSC (Desired State Configuration) resources using dsc and dsc_lite.
- Reboot Windows as part of management as necessary through reboot.
- Enforce fine-grained access control permissions using acl.
- Manage Windows Server Update Service configs on client nodes wsus_client.
- Install or remove Windows features with windowsfeature.
- Download files for use during management via download_file.
- Build IIS sites and virtual applications with iis.
- Install packages with chocolatey.
- Manage environment variables with windows_env.

### Install the Windows module pack

These steps show you how to install the module pack locally, but you can also install it on the master and pluginsync will push the module pack to all of your nodes.

1. Open the Puppet command prompt. If you haven't opened the command line interface before, enter Command Prompt Puppet in your Start Menu, and click Start Command Prompt with Puppet.
2. To list modules that you currently have installed, enter puppet module list in your Command Prompt window. If you're just getting started, you likely have no modules installed yet.
3. Next, to install the puppetlabs/windows module pack, type puppet module install puppetlabs/windows.

Notice that you get a nice output of everything that’s installed.

```
C:\>puppet module install puppetlabs/windows
Notice: Preparing to install into
C:/ProgramData/PuppetLabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
C:/ProgramData/PuppetLabs/code/environments/production/modules
### puppetlabs-windows (v2.1.0)
### chocolatey-chocolatey (v1.2.0)
# ### badgerious-windows_env (v2.2.2)
### puppet-download_file (v1.2.1)
### puppet-iis (v1.4.1)
### puppet-windowsfeature (v1.1.0)
### puppetlabs-acl (v1.1.1)
### puppetlabs-powershell (v1.0.5)
### puppetlabs-reboot (v1.2.0)
### puppetlabs-registry (v1.1.2)
# ### puppetlabs-stdlib (v4.9.0)
### puppetlabs-wsus_client (v1.0.0)
```

### Manage permissions with acl

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, you’ll 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

```
acI { 'c:\Users' :
  inherit_parent_permissions => 'false',
  permissions => [ 
    {identity => 'SYSTEM', rights=> ['full']},
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute'], affects => 'self_only'},
    {identity => 'Users', rights => ['read', 'execute'], affects => 'children_only'},
    {identity => 'Everyone', rights => ['read', 'execute'], affects => 'self_only'},
    {identity => 'Everyone', rights => ['read', 'execute'], affects => 'children_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 will have 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 will likely 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
file{'c:/tempperms':
  ensure => directory,
}

# By default, the acl will create 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`.

```perl
acl {'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
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 will happily manage 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:

```perl
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:

```plaintext
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' } ] to [ { identity => 'BUILTIN\Administrators', rights => ['full'] }, { identity => 'BUILTIN\Users', rights => ['read', 'execute'] } ]
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`

```plaintext
C:\>icacls c:\tempperms
c:\tempperms BUILTIN\Administrators:(OI)(CI)(F)
   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 Puppet managed registry keys with registry**

Eventually, you will likely 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'`

```plaintext
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'

```plaintext
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:

```plaintext
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.
```plaintext
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:
```plaintext
registry_key { '32:HKLM\Software\zTemporaryPuppet':
    ensure => present,
}
```

```plaintext
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.

![Registry Editor](image)

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](image)

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:

```puppet
class puppetconf::disable_error_reporting {
```

© 2020 Puppet, Inc.
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:

```plaintext
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:

```yaml
class {'chocolatey':
  chocolatey_download_url => 'https://internalurl/to/chocolatey.nupkg',
  use_7zip => false,
  log_output => true,
}
```

**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:

```yaml
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.

```plaintext
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 master 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.

```perl
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',
}

package {'roundhouse':
    ensure => '0.8.5.0',
}

class {'chocolatey_server':
    server_package_source => 'C:/packages',
}

package {'launchy':
    ensure => 'installed',
    install_options => ['-override', '-installArgs','"/VERYSILENT","/NORESTART",""'],
}

package {'git':
    ensure => 'latest',
}
```

**Copy an existing package and make it internal (repackaging packages)**

To make the existing package local, use these steps.

Chocolatey'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, as they will be regenerated (and their existence leads to issues).

```
notepadplusplus.commandline.6.8.7.nupkg
###_rels  # DELETE
###package  # DELETE
#  ###services
###tools
### [Content_Types].xml  # DELETE
### notepadplusplus.commandline.nuspec
```

4. Open tools\chocolateyInstall.ps1.

```
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).

```
$toolsDir = "$(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 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/02c21bebe5aab495a56747cbb9b4b5415c933fc0/icons/notepadplusplus.png</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Notepad++ is a free (as in "free speech" and also as in "free beer") source code editor and Notepad replacement that supports several languages. </description>
    <tags>notepad notepadplusplus notepad-plus-plus</tags>
  </metadata>
</package>
```

Some organizations will change the version field to denote 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>\notepadplusplus.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.

11. Update the manifest (chocolatey.pp) to use the custom package.

    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. The important thing you should do when creating a package is match the version of the package in this nuspec file to the version of the underlying software as closely as possible. You will package Vagrant into this package you’ve created, so the version of the package in the nuspec file should match. 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 will also download 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, you will 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.

```puppet
package {'vagrant':
    ensure => installed,
    provider => chocolatey,
    source => 'c:\packages',
}
```

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.

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 will be 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:

You should see a similar success message:

Enabled autoUninstaller

2. To remove Vagrant, edit your chocolatey.pp manifest to ensure => absent. Then save and validate the file.

```yaml
package {'vagrant':
  ensure => absent,
  provider => chocolatey,
  source => 'c:\packages',
}
```

3. Next, run puppet apply <FILE PATH>\chocolatey.pp to apply the manifest.

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!

## 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 compile masters and ActiveMQ hubs and spokes. You can install agents on *nix, Windows, and macOS nodes, and on certain network devices.
Choosing an architecture

There are several configurations available for Puppet Enterprise. The configuration you use depends on the number of nodes in your environment, the resources required to serve your agent catalogs, and your availability requirements.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Node limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic installation</td>
<td>The master, console, and PuppetDB are all installed on one node. This installation type is the easiest to install, upgrade, and troubleshoot.</td>
<td>Up to several thousand</td>
</tr>
<tr>
<td>(Recommended)</td>
<td>Tip: You can add high availability to this configuration.</td>
<td></td>
</tr>
</tbody>
</table>

System requirements on page 148
Refer to these system requirements for Puppet Enterprise installations.

What gets installed and where? on page 163
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.

Installing Puppet Enterprise on page 172
You can install PE in a monolithic configuration, where all infrastructure components are installed on one node, or in a split configuration, where the master, PuppetDB, and console are installed on separate nodes.

Purchasing and installing a license key on page 186
You can download and install Puppet Enterprise on 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, PE logs license warnings until you install an appropriate license key.

Installing agents on page 186
You can install Puppet Enterprise agents on *nix, Windows, and macOS.

Installing network device agents on page 202
Install agents on network switches to operate them with Puppet Enterprise as managed devices.

Installing compile masters on page 205
As your Puppet Enterprise infrastructure scales up to 4,000 nodes and beyond, add load-balanced compile masters to your monolithic installation to increase the number of agents you can manage.

Installing ActiveMQ hubs and spokes on page 209
Add hubs and spokes to large Puppet Enterprise deployments for efficient load balancing and for relaying MCollective messages.

Installing PE client tools on page 213
PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily managed by Puppet.

Installing external PostgreSQL on page 217
By default, Puppet Enterprise includes its own database backend, PE-PostgreSQL, which is installed alongside PuppetDB. If the load on your PuppetDB node is larger than it can effectively scale to (greater than 20,000 nodes), you can install a standalone instance of PE-PostgreSQL.

Uninstalling on page 221
Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Node limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic installation with compile masters</td>
<td>The master of masters, console, and PuppetDB are installed on one node, and one or more compile masters help distribute the agent catalog compilation workload.</td>
<td>Less than 20,000</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> You can add high availability to this configuration.</td>
<td></td>
</tr>
<tr>
<td>Split installation</td>
<td>The master, console, and PuppetDB are each installed on separate nodes. Use this installation type only if you have a limit on the number of permitted cores per server, or if you are managing more than 20,000 nodes.</td>
<td>More than 20,000</td>
</tr>
<tr>
<td>Large environment installation</td>
<td>The master of masters, console, and PuppetDB are each installed on separate nodes, and one or more compile masters help distribute the agent catalog compilation workflow.</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>

**Monolithic installation**

![Diagram of Monolithic installation](image)

- **End Users**
- **Puppet Master** (includes PuppetDB and Puppet console)
- **Puppet Agents**
Monolithic installation with compile masters

Split installation
Large environment installation

System requirements

Refer to these system requirements for Puppet Enterprise installations.

- **Hardware requirements** on page 148
  These hardware requirements are based on internal testing at Puppet and are meant only as guidelines to help you determine your hardware needs.
- **Supported operating systems** on page 152
  Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.
- **Supported browsers** on page 157
  The following browsers are supported for use with the console.
- **System configuration** on page 158
  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 meant only as guidelines to help you determine your hardware needs.

Your installation's existing code base can significantly affect performance, so adjusting expectations based on differences between your installation's code base and the testing code base as documented in How we develop hardware requirements is important to make the best use of the requirements on this page.

Monolithic hardware requirements

In monolithic installations, hardware requirements differ depending on the number of nodes you're managing. To take advantage of progressively larger hardware, you must configure your environment to use additional resources.

<table>
<thead>
<tr>
<th>Node volume</th>
<th>Cores</th>
<th>RAM</th>
<th>/opt/</th>
<th>/var/</th>
<th>AWS EC2</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial use</td>
<td>2</td>
<td>8 GB</td>
<td>20 GB</td>
<td>24 GB</td>
<td>m5.large</td>
<td>A2 v2</td>
</tr>
<tr>
<td>Up to 2,000</td>
<td>4</td>
<td>8 GB</td>
<td>50 GB</td>
<td>24 GB</td>
<td>c5.xlarge</td>
<td>F4s v2</td>
</tr>
<tr>
<td>Up to 3,500</td>
<td>8</td>
<td>16 GB</td>
<td>80 GB</td>
<td>24 GB</td>
<td>c5.2xlarge</td>
<td>F8s v2</td>
</tr>
<tr>
<td>Up to 4,000</td>
<td>16</td>
<td>32 GB</td>
<td>100 GB</td>
<td>24 GB</td>
<td>c5.4xlarge</td>
<td>F16s v2</td>
</tr>
</tbody>
</table>

Important notes on the data in this chart:
• **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.

• **Azure:** Azure requirements are not currently tested by Puppet, but are presented here as our best guidance based on comparable EC2 instance testing.

• **/opt/ storage requirements:** The database should not exceed 50% of /opt/ to allow for future upgrades.

• **/var/ storage requirements:** There are roughly 20 log files stored in /var/ which are limited in size to 1 GB each. Log retention settings make it unlikely that the maximum capacity will be needed, but we recommend allocating 24 GB to avoid issues.

**Related information**

**Methods for configuring Puppet Enterprise** on page 235

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 own certificate to the whitelist, increase the max-threads setting for `http` and `https` requests, or configure the number of JRuby instances.

**Tuning monolithic installations** on page 254

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

**Monolithic with compile masters hardware requirements**

If you are managing more than 4,000 nodes, you can add load-balanced compile masters to your monolithic installation to increase the number of agents you can manage.

Each compile master increases capacity by approximately 1,500–3,000 nodes, until you exhaust the capacity of PuppetDB or the console, which run on the master of masters. If you start to see performance issues around 8,000 nodes, you can adjust your hardware or move to a larger base infrastructure.

**Note:** When you expand your deployment to use compile masters, you must also start using load balancers. It is simpler to upgrade your hardware in your monolithic installation, if you can, than to add compile masters and load balancers.

To manage more than 4,000 nodes, we recommend the following minimum hardware:

<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>4,000–20,000</td>
<td>Monolithic node</td>
<td>16</td>
<td>32</td>
<td>150</td>
<td>10</td>
<td>c5.4xlarge</td>
</tr>
<tr>
<td></td>
<td>Each compiler (1,500 - 3,000 nodes)</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>2</td>
<td>m5.xlarge</td>
</tr>
</tbody>
</table>

**Note:** If you need to go beyond 20,000 nodes contact Support or your sales team before setting up a large environment installation.

**Related information**

**Methods for configuring Puppet Enterprise** on page 235

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 own certificate to the whitelist, increase the max-threads setting for `http` and `https` requests, or configure the number of JRuby instances.

**Tuning monolithic installations** on page 254

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

**Large environment hardware requirements**

A large environment installation is a high-capacity infrastructure. It runs on a split installation with additional compile masters and ActiveMQ message brokering. This installation is suitable for managing more than 20,000 nodes.

With this installation type, you can support more nodes by adding more resources to PuppetDB and increasing the number of compile masters.

© 2020 Puppet, Inc.
<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>over 20,000</td>
<td>master</td>
<td>16</td>
<td>32</td>
<td>150</td>
<td>10</td>
<td>c4.4xlarge</td>
</tr>
<tr>
<td></td>
<td>console</td>
<td>4</td>
<td>16</td>
<td>30</td>
<td>2</td>
<td>m3.xlarge</td>
</tr>
<tr>
<td></td>
<td>PuppetDB</td>
<td>32</td>
<td>48</td>
<td>200</td>
<td>70</td>
<td>m3.2xlarge</td>
</tr>
<tr>
<td></td>
<td>compile master</td>
<td>4</td>
<td>16</td>
<td>30</td>
<td>42</td>
<td>m3.xlarge or m4.xlarge</td>
</tr>
<tr>
<td></td>
<td>ActiveMQ hubs</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>18</td>
<td>m3.large instance</td>
</tr>
<tr>
<td></td>
<td>ActiveMQ spokes</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>18</td>
<td>m3.large</td>
</tr>
</tbody>
</table>

**How we develop hardware requirements: Performance test methods**

Puppet tests the performance of Puppet Enterprise and develops hardware requirements based on our testing. We continue to invest in improvements in our performance testing methods.

The performance of Puppet Enterprise is highly dependent on the code base used by a specific installation. The details included here are intended to help you evaluate how your specific installation compares to what we used to generate our scale and hardware estimates so that you can better estimate what you need to manage your Puppet infrastructure.

**Tools: Gatling and puppet infrastructure tune**

We use gatling-puppet-load-test, a framework we’ve developed to perform load and performance testing for PE and open source Puppet. It uses Gatling, an open source load and performance testing tool that records and replays HTTP traffic, then generates reports about the performance of the simulated requests. Using Gatling allows us to simulate agent requests in a full PE installation where Puppet Server is driving communication with PuppetDB, the node classifier, and other tools.

We also use puppet infrastructure tune, which provides configuration settings to apply to the installation.

**Platform: Amazon EC2**

We use Amazon EC2 instances for our performance tests with a master node running PE and a metrics node running the simulated agents and capturing the performance data. We know that the instances’ performance varies, but this has less of an impact on overall performance than the variations in the size and complexity of users' code bases.

**Table 1: EC2 instance types**

<table>
<thead>
<tr>
<th>AWS EC2</th>
<th>Cores</th>
<th>RAM</th>
<th>Testing use</th>
</tr>
</thead>
<tbody>
<tr>
<td>m5.large</td>
<td>2</td>
<td>8 GB</td>
<td>Confirming trial size works</td>
</tr>
<tr>
<td>c5.xlarge</td>
<td>4</td>
<td>8 GB</td>
<td>Scale</td>
</tr>
<tr>
<td>c5.2xlarge</td>
<td>8</td>
<td>16 GB</td>
<td>Scale, Apples-to-apples, Soak</td>
</tr>
<tr>
<td>c5.4xlarge</td>
<td>16</td>
<td>32 GB</td>
<td>Scale</td>
</tr>
</tbody>
</table>

**Test environment**

**Control repository**

We use puppetlabs-pe_perf_control_repo with r10k to classify the simulated agents using roles and profiles, with three sizes defined.

**Facts and reports**

© 2020 Puppet, Inc.
Although the simulated agents all share the same classification, we use dynamic facts and varying report responses (no change, failure, intentional change, corrective change) to better simulate a typical customer environment.

<table>
<thead>
<tr>
<th>Resources</th>
<th>role::by_size::small</th>
<th>role::by_size::medium</th>
<th>role::by_size::large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>129</td>
<td>750</td>
<td>1267</td>
</tr>
<tr>
<td>Classes</td>
<td>24</td>
<td>71</td>
<td>120</td>
</tr>
<tr>
<td>Facts</td>
<td>151</td>
<td>151</td>
<td>151</td>
</tr>
</tbody>
</table>

**Classification**

```ruby
class role::by_size::small {
  include ::profile::tomcat::basic
  include ::profile::postgresql::basic
}
class role::by_size::medium {
  include ::role::by_size::small

  include ::profile::users
  include ::profile::sysop::packages
  include ::profile::motd
  include ::profile::hiera_check
}
class role::by_size::large {
  include ::role::by_size::medium

  include ::profile::apache::basic
  include ::profile::influxdb::basic
  include ::profile::loop_through_file_resources
}
```

**Environment cache**

Although we recommend enabling environment caching to improve performance in production environments, we do not enable it in our performance testing environment to avoid unrealistic results due to the identical classification of the simulated agents.

**Testing methodology**

**Warming up Puppet Server**

JRuby 9000 provides better overall performance than the previous version, but it also comes with a warm-up burden. For some tests we want the Puppet Server process to be warmed up before we begin. Our warm-up process involves running a portion of the agents to generate about 300 connections to the endpoints served by JRubies, multiplied by the number of configured JRuby instances. In general, the closer you are to the node capacity of your install, the faster it will warm up when running the normal load. Our process takes about five minutes.

**Apples-to-apples testing**

To do an apples-to-apples test, we run a simulation for two different PE versions of 600 agents each (classified as large) checking in at the default 30-minute interval for eight iterations. We run the warm-up procedure before the simulation. Then we compare response times, system resource usage, and process resource usage. We do this on a weekly basis for all active development branches to ensure that general performance doesn’t degrade during development.

**Scale testing**

In order to determine the node capacity of each targeted EC2 instance type, we perform scale testing with a small load that incrementally increases the number of simulated agents until requests begin to time out or respond with errors.
The scale test is structured to simulate the default 30-minute agent check-in interval. We start the test with an agent volume below the expected performance threshold and add 100 agents at a time until more than 10 timeouts are encountered. For example, when testing the EC2 c5.2xlarge (8 cores and 16 GB), we start with 3,000 agents.

**Cold-start versus warm-start scale testing**

To simulate recovering from a Puppet Server restart, we use a cold-start scenario. We restart the `pe-puppetserver` service between each iteration so that the warm-up burden will reduce the maximum node count. This is the most conservative estimate of node capacity we test for.

To determine burst capacity, we use a warm-start scenario. When doing a warm-start scale test, we don’t restart the `pe-puppetserver` service, and we run the test on the same host that ran the cold-start tests to ensure it is already warmed up.

The extra capacity provided by a warmed up system is great for serving Puppet agent runs triggered by orchestration from the console, Continuous Delivery for Puppet Enterprise, or Bolt. Therefore we recommend keeping your node count near the cold-start numbers.

**Soak testing**

To ensure PE doesn’t suffer a performance degradation over time we run a soak test on each release. We use 600 agents, classified as large, checking in at the default 30-minute interval for 14 days. We run the warm-up before we start the test. This two-week test lets us verify that all garbage collection features are being triggered and keeping things under control. We check the Gatling response time graphs to ensure that performance is stable throughout the test.

**Supported operating systems**

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

**Supported operating systems and devices**

When choosing an operating system, first consider the machine’s role. Different roles support different operating systems and architectures.

**Tip:** For details about platform support lifecycles and planned end-of-life support, see [Platform support lifecycle](#) on the Puppet website.

**Master platforms**

The master role can be installed on these operating systems and architectures.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Linux</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>• CentOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Oracle Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Red Hat Enterprise Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Scientific Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>12</td>
<td>x86_64</td>
</tr>
<tr>
<td>Ubuntu (General Availability kernels)</td>
<td>16.04, 18.04</td>
<td>amd64</td>
</tr>
</tbody>
</table>

**Agent platforms**

The agent role can be installed on these operating systems and architectures.

<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></td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Linux</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Debian</td>
<td>Jessie (8), Stretch (9), Buster (10)</td>
<td>• i386</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• amd64</td>
</tr>
<tr>
<td>Enterprise Linux</td>
<td>5, 6, 7, 8</td>
<td>• x86_64</td>
</tr>
<tr>
<td>• CentOS</td>
<td></td>
<td>• i386 for versions 5, 6</td>
</tr>
<tr>
<td>• Oracle Linux</td>
<td></td>
<td>• ppc64le for version 7</td>
</tr>
<tr>
<td>• Red Hat Enterprise Linux</td>
<td></td>
<td>• aarch64 for version 7</td>
</tr>
<tr>
<td>• Scientific Linux</td>
<td></td>
<td>• FIPS 140-2 compliant Enterprise Linux 7</td>
</tr>
<tr>
<td>Fedora</td>
<td>28, 29, 30, 31</td>
<td>• x86_64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• i386 for version 25</td>
</tr>
<tr>
<td>macOS</td>
<td>10.12, 10.13, 10.14</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>7, 8.1, 10</td>
<td>• x64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• x86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• x86 for 2008</td>
</tr>
<tr>
<td>Solaris</td>
<td>10, 11</td>
<td>• SPARC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• i386</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>11, 12, 15</td>
<td>• x86_64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• i386 for version 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ppc64le for version 12</td>
</tr>
<tr>
<td>Ubuntu (General Availability kernels)</td>
<td>14.04, 16.04, 18.04</td>
<td>• amd64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• i386 for versions 14.04 and 16.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ppc64el for version 16.04</td>
</tr>
</tbody>
</table>

**Note:** Some operating systems require an active subscription with the vendor's package management system (for example, the Red Hat Network) to install dependencies.

**Agent network devices**

The agent role can be installed on these network devices and operating systems. Refer to the Forge for specific system requirements and dependencies.

- Arista EOS
- Cisco IOS-XR
- Cisco NX-OS
- Cumulus Linux

© 2020 Puppet, Inc.
CentOS dependencies
When you install PE or an agent, these packages are also installed from the various operating system repositories.
If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

<table>
<thead>
<tr>
<th></th>
<th>All Nodes</th>
<th>Master Nodes</th>
<th>Console Nodes</th>
<th>Console/Console DB Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pciutils</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system-logos</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>which</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dmidecode</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>net-tools</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>curl</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>mailcap</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libjpeg</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libtool-ltdl</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unixODBC</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libxslt</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>zlib</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

RHEL dependencies
When you install PE or an agent, these packages are also installed from the various operating system repositories.
If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

<table>
<thead>
<tr>
<th></th>
<th>All Nodes</th>
<th>Master Nodes</th>
<th>Console Nodes</th>
<th>Console/Console DB Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pciutils</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system-logos</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>which</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libxml2</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dmidecode</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>net-tools</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cronie (RHEL 6, 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vixie-cron (RHEL 4, 5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>curl</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>mailcap</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libjpeg</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libtool-ltdl (RHEL 7)</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
### SUSE Linux Enterprise Server dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

<table>
<thead>
<tr>
<th></th>
<th>All Nodes</th>
<th>Master Nodes</th>
<th>Console Nodes</th>
<th>Console/Console DB Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>unixODBC (RHEL 7)</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libxslt</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>gtk2</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Ubuntu dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn’t have internet access, you must manually install dependencies.

<table>
<thead>
<tr>
<th></th>
<th>All Nodes</th>
<th>Master Nodes</th>
<th>Console Nodes</th>
<th>Console/Console DB Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pciutils</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pmtools</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cron</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libxml2</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>net-tools</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libxslt</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>curl</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>libjpeg</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>db43</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>unixODBC</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>zlib</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>All Nodes</td>
<td>Master Nodes</td>
<td>Console Nodes</td>
<td>Console/Console DB Nodes</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>---------------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td>libmagic1</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libpcre3</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>curl</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>perl</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mime-support</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libcap2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libjpeg62</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libxml1.1</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libgtk2.0-0</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ca-certificates-java</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>openjdk-7-jre-headless*</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>libossp-uuid16</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>zlib</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*For Ubuntu 18.04, use openjdk-8-jre-headless. This package requires the universe repository to install.

### AIX dependencies and limitations

Before installing the agent on AIX systems, install these packages.

- bash
- zlib
- readline
- curl
- OpenSSL

**CAUTION:** For cURL and OpenSSL, 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.


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.

© 2020 Puppet, Inc.
Solaris dependencies and limitations
Solaris support is agent only.

For Solaris 10, these packages are required:

• 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:

• system/readline
• system/library/gcc-45-runtime
• library/security/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 CentOS 6.0 to 7.0; it does not refer to a minor version upgrade, like CentOS 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 –c uninstaller options).
3. Upgrade your operating system.
4. Install PE.
5. Restore your backup.

Related information
Back up your PE infrastructure on page 758
PE backup creates a copy of your Puppet infrastructure, including configuration, certificates, code, and PuppetDB.

Restore your Puppet Enterprise infrastructure on page 759
Use the restore commands to migrate your PE master to a new host or to recover from system failure.

Uninstalling on page 221
Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

Installing Puppet Enterprise on page 172
You can install PE in a monolithic configuration, where all infrastructure components are installed on one node, or in a split configuration, where the master, PuppetDB, and console are installed on separate nodes.

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>Microsoft Internet Explorer</td>
<td>11</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.

**Timekeeping and name resolution**
Before installing, there are network requirements you need to consider and prepare for. The most important requirements include syncing time and creating a plan for name resolution.

**Timekeeping**
Use NTP or an equivalent service to ensure that time is in sync between your master, 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 master. Ensure that the master 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 master reachable at the hostname `puppet`, which is the default master 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.

<table>
<thead>
<tr>
<th>URL</th>
<th>Enables</th>
</tr>
</thead>
<tbody>
<tr>
<td>forgeapi.puppet.com</td>
<td>Puppet module downloads.</td>
</tr>
<tr>
<td>pm.puppetlabs.com</td>
<td>Agent module package downloads.</td>
</tr>
<tr>
<td>s3.amazonaws.com</td>
<td>Agent module package downloads (redirected from pm.puppetlabs.com).</td>
</tr>
<tr>
<td>rubygems.org</td>
<td>Puppet and Puppet Server gem downloads.</td>
</tr>
<tr>
<td>github.com</td>
<td>Third-party module downloads not served by the Forge and access to control repositories.</td>
</tr>
</tbody>
</table>
Firewall configuration for monolithic installations
These are the port requirements for monolithic installations.

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8140  | • The master uses this port to accept inbound traffic/requests from agents.  
• The console sends requests to the master on this port.  
• Certificate requests are passed over this port unless ca_port is set differently.  
• status checks are sent over this port.  
• Classifier group: **PE Master** |
| 443   | • This port provides host access to the console  
• The console accepts HTTPS traffic from end users on this port.  
• Classifier group: **PE Console** |
| 61613 | • uses this port to accept inbound traffic/requests from agents.  
• Any host used to invoke commands must be able to reach on this port.  
• Classifier group: **PE ActiveMQ Broker** |
| 8142  | • Orchestrator and the Run Puppet button use this port on the master of masters to accept inbound traffic/responses from agents via the agent.  
• Classifier group: **PE Orchestrator** |
## Firewall configuration for monolithic installations with compile masters

These are the port requirements for monolithic installations with compile masters.

<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8140  | • The master uses this port to accept inbound traffic/requests from agents.  
      | • The console sends requests to the master on this port.  
      | • Certificate requests are passed over this port unless `ca_port` is set differently.  
      | • Status checks are sent over this port.  
      | • The master uses this port to send status checks to compile masters. (Not required to run PE.)  
      | • Classifier group: **PE Master** |
| 443   | • This port provides host access to the console  
      | • The console accepts HTTPS traffic from end users on this port.  
      | • Classifier group: **PE Console** |
| 61613 | • uses this port to accept inbound traffic/requests from agents.  
      | • Any host used to invoke commands must be able to reach on this port.  
      | • Classifier group: **PE ActiveMQ Broker** |
| 4433  | • This port is used as a classifier / console services API endpoint.  
      | • The master communicates with the console over this port.  
      | • Classifier group: **PE Console** |
| 8081  | • accepts traffic/requests on this port.  
      | • The master and console send traffic to on this port.  
      | • Status checks are sent over this port.  
<pre><code>  | • Classifier group: **PE PuppetDB** |
</code></pre>
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8142  | • Orchestrator and the **Run Puppet** button use this port on the master of masters to accept inbound traffic/ responses from agents via the agent.  
      • Classifier group: **PE Orchestrator** |
| 8143  | • Orchestrator uses this port to accept connections from brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the master of masters. If you install the client on a workstation, this port must be available on the workstation.  
      • Classifier group: **PE Orchestrator** |

**Firewall configuration for split installations**

These are the port requirements for split installations.
<table>
<thead>
<tr>
<th>Port</th>
<th>Use</th>
</tr>
</thead>
</table>
| 8140  | • The master uses this port to accept inbound traffic/requests from agents.  
       | • The console sends requests to the master on this port.              
       | • Certificate requests are passed over this port unless `ca_port` is set differently. |
       | • status checks are sent over this port.                             
       | • In a large environment installation, the master uses this port to send status checks to compile masters. (Not required to run PE.) |
       | • Classifier group: **PE Master**                                    |
| 443   | • This port provides host access to the console                     
       | • The console accepts HTTPS traffic from end users on this port.     |
       | • Classifier group: **PE Console**                                   |
| 61613 | • uses this port to accept inbound traffic/requests from agents.     
       | • Any host used to invoke commands must be able to reach on this port.|
       | • Classifier group: **PE ActiveMQ Broker**                           |
| 4433  | • This port is used as a classifier / console services API endpoint.  
       | • The master communicates with the console over this port.           |
       | • Classifier group: **PE Console**                                   |
| 8081  | • accepts traffic/requests on this port.                            
       | • The master and console send traffic to on this port.               
       | • status checks are sent over this port.                            |
       | • Classifier group: **PE PuppetDB**                                  |
| 8142  | • Orchestrator and the **Run Puppet** button use this port on the master of masters to accept inbound traffic/responses from agents via the agent. |
       | • Classifier group: **PE Orchestrator**                              |
| 8143  | • Orchestrator uses this port to accept connections from brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the master of masters. If you install the client on a workstation, this port must be available on the workstation. |
       | • Classifier group: **PE Orchestrator**                              |
| 5432  | • PostgreSQL runs on this port.                                     
       | • The console node needs to connect to the PuppetDB node hosting the PostgreSQL database on this port. |
       | • Classifier group: **PE PuppetDB**                                  |
Port usage for all installation types
In addition to installation-specific firewall configuration, some features and tools have specific port requirements.

- Port **3000**: If you are installing using the web-based installer, in either a split or a mono configuration, ensure port 3000 is open. You can close this port when the installation is complete. Instructions for port forwarding to the web-based installer are included in the installation instructions.
- Port **8150** and **8151**: Razor uses port 8150 for HTTP and 8151 for HTTPS. Any node classified as a Razor server must be able to use these ports.
- Port **4432**: Local connections for the node classifier, activity service, and RBAC status checks are sent over this port. Remote connections should use port **4433**.
- Port **8170**: Code Manager uses this port to deploy environments, run webhooks, and make API calls.

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 releases dating back to the previous long term supported (LTS) release.

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**: 5 is a backwards-compatible evolution of 4, which is built into 4.9.0 and higher. To provide some backwards-compatible features, it uses the classic 3.x.x codebase version listed in this table.
<table>
<thead>
<tr>
<th>Version</th>
<th>Agent</th>
<th>Agent</th>
<th>Version</th>
<th>Agent</th>
<th>Agent</th>
<th>Ruby</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.16</td>
<td>5.5.21</td>
<td>5.5.21</td>
<td>3.11.14</td>
<td>3.4.6</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.1.1g</td>
</tr>
<tr>
<td>2018.15</td>
<td>5.5.20</td>
<td>5.5.20</td>
<td>3.11.13</td>
<td>3.4.6</td>
<td>2.12.5</td>
<td>2.4.10</td>
<td>1.0.2u</td>
</tr>
<tr>
<td>2018.13</td>
<td>5.5.19</td>
<td>5.5.19</td>
<td>3.11.12</td>
<td>3.4.6</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2u</td>
</tr>
<tr>
<td>2018.12</td>
<td>5.5.18</td>
<td>5.5.18</td>
<td>3.11.11</td>
<td>3.4.6</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2t</td>
</tr>
<tr>
<td>2018.11</td>
<td>5.5.17</td>
<td>5.5.17</td>
<td>3.11.10</td>
<td>3.4.6</td>
<td>2.12.5</td>
<td>2.4.9</td>
<td>1.0.2t</td>
</tr>
<tr>
<td>2018.10</td>
<td>5.5.16</td>
<td>5.5.16</td>
<td>3.11.9</td>
<td>3.4.6</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2t</td>
</tr>
<tr>
<td>2018.09</td>
<td>5.5.14</td>
<td>5.5.14</td>
<td>3.11.8</td>
<td>3.4.6</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2r</td>
</tr>
<tr>
<td>2018.08</td>
<td>5.5.10</td>
<td>5.5.10</td>
<td>3.11.7</td>
<td>3.4.6</td>
<td>2.12.4</td>
<td>2.4.5</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.07</td>
<td>5.5.8</td>
<td>5.5.8</td>
<td>3.11.6</td>
<td>3.4.5</td>
<td>2.12.4</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.06</td>
<td>5.5.6</td>
<td>5.5.6</td>
<td>3.11.4</td>
<td>3.4.4</td>
<td>2.12.3</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.05</td>
<td>5.5.4</td>
<td>5.5.3</td>
<td>3.11.3</td>
<td>3.4.3</td>
<td>2.12.2</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.04</td>
<td>5.5.3</td>
<td>5.5.2</td>
<td>3.11.2</td>
<td>3.4.3</td>
<td>2.12.2</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2018.03</td>
<td>5.5.1</td>
<td>5.5.1</td>
<td>3.11.1</td>
<td>3.4.3</td>
<td>2.12.1</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.10</td>
<td>5.3.8</td>
<td>5.3.7</td>
<td>3.9.6</td>
<td>3.4.3</td>
<td>2.11.5</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.09</td>
<td>5.3.8</td>
<td>5.3.7</td>
<td>3.9.6</td>
<td>3.4.3</td>
<td>2.11.5</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.08</td>
<td>5.3.6</td>
<td>5.3.6</td>
<td>3.9.6</td>
<td>3.4.3</td>
<td>2.11.5</td>
<td>2.4.4</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.07</td>
<td>5.3.5</td>
<td>5.3.5</td>
<td>3.9.5</td>
<td>3.4.2</td>
<td>2.11.4</td>
<td>2.4.3</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.06</td>
<td>5.3.4</td>
<td>5.3.4</td>
<td>3.9.4</td>
<td>3.4.2</td>
<td>2.11.4</td>
<td>2.4.3</td>
<td>1.0.2n</td>
</tr>
<tr>
<td>2017.05</td>
<td>5.3.3</td>
<td>5.3.3</td>
<td>3.9.3</td>
<td>3.4.2</td>
<td>2.11.4</td>
<td>2.4.2</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.04</td>
<td>5.3.2</td>
<td>5.3.2</td>
<td>3.9.2</td>
<td>3.4.2</td>
<td>2.11.3</td>
<td>2.4.1</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.03</td>
<td>5.3.2</td>
<td>5.3.2</td>
<td>3.9.2</td>
<td>3.4.2</td>
<td>2.11.3</td>
<td>2.4.1</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.02</td>
<td>1.10.9</td>
<td>4.10.9</td>
<td>3.6.8</td>
<td>3.3.2</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.01</td>
<td>1.10.8</td>
<td>4.10.8</td>
<td>3.6.7</td>
<td>3.3.2</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.00</td>
<td>1.10.5</td>
<td>4.10.5</td>
<td>3.6.6</td>
<td>3.3.2</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.02</td>
<td>1.10.4</td>
<td>4.10.4</td>
<td>3.6.5</td>
<td>3.3.2</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.01</td>
<td>1.10.1</td>
<td>4.10.1</td>
<td>3.6.4</td>
<td>3.3.1</td>
<td>2.10.4</td>
<td>2.1.9</td>
<td>1.0.2k</td>
</tr>
<tr>
<td>2017.00</td>
<td>1.9.3</td>
<td>4.9.4</td>
<td>3.6.2</td>
<td>3.3.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td>1.0.2j</td>
</tr>
<tr>
<td>2017.00 for Ubuntu on i386</td>
<td>1.7.0</td>
<td>4.7.0</td>
<td>3.4.1</td>
<td>3.2.1</td>
<td>2.10.2</td>
<td>2.1.9</td>
<td>1.0.2h</td>
</tr>
<tr>
<td>2017.00 for Ubuntu on i386</td>
<td>1.7.0</td>
<td>4.7.0</td>
<td>3.4.1</td>
<td>3.2.1</td>
<td>2.10.2</td>
<td>2.1.9</td>
<td>1.0.2h</td>
</tr>
<tr>
<td>Version</td>
<td>Agent</td>
<td>Java</td>
<td>Nginx</td>
<td>Ruby</td>
<td>OpenSSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.5.2</td>
<td>1.8.3</td>
<td>3.2.2</td>
<td>2.9.1</td>
<td>2.1.9</td>
<td>1.0.2j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.5.2 for Ubuntu on i386</td>
<td>1.7.0</td>
<td>3.2.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td>1.0.2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.5.1</td>
<td>1.8.2</td>
<td>3.2.2</td>
<td>2.9.1</td>
<td>2.1.9</td>
<td>1.0.2j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.5.1 for Ubuntu on i386</td>
<td>1.7.0</td>
<td>3.2.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td>1.0.2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.15</td>
<td>1.10.14</td>
<td>3.3.3</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td>1.0.2n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.14</td>
<td>1.10.14</td>
<td>3.3.3</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td>1.0.2n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.13</td>
<td>1.10.14</td>
<td>3.3.3</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td>1.0.2n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.11</td>
<td>1.10.12</td>
<td>3.3.3</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td>1.0.2n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.10</td>
<td>1.10.10</td>
<td>3.6.9</td>
<td>N/A</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.9</td>
<td>1.10.9</td>
<td>3.6.8</td>
<td>N/A</td>
<td>2.10.6</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.8</td>
<td>1.10.8</td>
<td>3.6.7</td>
<td>N/A</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.7</td>
<td>1.10.5</td>
<td>3.6.6</td>
<td>N/A</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.6</td>
<td>1.10.4</td>
<td>3.6.5</td>
<td>N/A</td>
<td>2.10.5</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.5</td>
<td>1.10.1</td>
<td>3.6.4</td>
<td>N/A</td>
<td>2.10.4</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.3</td>
<td>1.7.2</td>
<td>3.4.2</td>
<td>3.2.2</td>
<td>2.9.1</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.3 for Ubuntu on i386</td>
<td>1.7.1</td>
<td>3.2.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td>1.0.2j</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.2</td>
<td>1.7.1</td>
<td>3.4.1</td>
<td>3.2.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.4.0</td>
<td>1.7.1</td>
<td>3.4.1</td>
<td>3.2.1</td>
<td>2.9.0</td>
<td>2.1.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This table shows components installed on server nodes:

<table>
<thead>
<tr>
<th>Version</th>
<th>Agent</th>
<th>Server</th>
<th>Java</th>
<th>Nginx</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.1.16</td>
<td>5.3.14</td>
<td>2.6.8</td>
<td>1.9.9</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.15</td>
<td>5.3.13</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.13</td>
<td>5.3.12</td>
<td>2.6.8</td>
<td>1.9.6</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.12</td>
<td>5.3.11</td>
<td>2.6.7</td>
<td>1.9.6</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.11</td>
<td>5.3.10</td>
<td>2.6.7</td>
<td>1.9.2</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.9</td>
<td>5.3.9</td>
<td>2.6.6</td>
<td>1.9.2</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.8</td>
<td>5.3.8</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.7</td>
<td>5.3.7</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.5</td>
<td>5.3.6</td>
<td>2.6.5</td>
<td>1.9.2</td>
<td>5.15.5</td>
</tr>
<tr>
<td>2018.1.4</td>
<td>5.3.5</td>
<td>2.6.2</td>
<td>1.9.2</td>
<td>5.15.3</td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Version</th>
<th>5.3.4</th>
<th>5.2.4</th>
<th>2.6.2</th>
<th>1.9.2</th>
<th>N/A</th>
<th>9.6.8</th>
<th>1.8.0</th>
<th>5.15.3</th>
<th>1.14.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.1.3</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>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</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>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.10</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.10</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.14.0</td>
</tr>
<tr>
<td>2017.3.9</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.8</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.14.0</td>
</tr>
<tr>
<td>2017.3.8</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.8</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.6</td>
<td>5.1.6</td>
<td>5.1.5</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.8</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.5</td>
<td>5.1.6</td>
<td>5.1.4</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.6</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.4</td>
<td>5.1.5</td>
<td>5.1.4</td>
<td>2.6.0</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.6</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.2</td>
<td>5.1.4</td>
<td>5.1.3</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.5</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.1</td>
<td>5.1.3</td>
<td>5.1.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.5</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.3.0</td>
<td>5.1.3</td>
<td>5.1.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.6.5</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.2.5</td>
<td>2.8.0</td>
<td>4.4.2</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.4.14</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.2.4</td>
<td>2.8.0</td>
<td>4.4.2</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.4.14</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.2.3</td>
<td>2.7.2</td>
<td>4.4.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.4.12</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2017.2.2</td>
<td>2.7.2</td>
<td>4.4.1</td>
<td>2.5.5</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.4.12</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.10.2</td>
</tr>
<tr>
<td>2017.2.1</td>
<td>2.7.2</td>
<td>4.4.0</td>
<td>2.5.4</td>
<td>1.6.0</td>
<td>3.1.2</td>
<td>9.4.10</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.10.2</td>
</tr>
<tr>
<td>2017.1.1</td>
<td>2.7.2</td>
<td>4.3.2</td>
<td>2.5.1</td>
<td>1.5.0</td>
<td>3.1.2</td>
<td>9.4.10</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.10.2</td>
</tr>
<tr>
<td>2017.1.0</td>
<td>2.7.2</td>
<td>4.3.2</td>
<td>2.5.1</td>
<td>1.5.0</td>
<td>3.1.2</td>
<td>9.4.10</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.10.2</td>
</tr>
<tr>
<td>2016.5.2</td>
<td>2.6.0</td>
<td>4.2.5</td>
<td>2.5.0</td>
<td>1.5.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.5.1</td>
<td>2.6.0</td>
<td>4.2.5</td>
<td>2.5.0</td>
<td>1.5.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.13.2</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.4.15</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.19</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.14.0</td>
</tr>
<tr>
<td>2016.4.14</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.17</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.14.0</td>
</tr>
<tr>
<td>2016.4.13</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.17</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.11</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.17</td>
<td>1.8.0</td>
<td>5.15.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.10</td>
<td>2.6.1</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.15</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.9</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.14</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.8</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.14</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.7</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.12</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.12.1</td>
</tr>
<tr>
<td>2016.4.6</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.12</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.4.5</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.5.4</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.4.3</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.5</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.14.3</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.4.2</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.3</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.13.2</td>
<td>1.8.1</td>
</tr>
<tr>
<td>2016.4.0</td>
<td>2.6.0</td>
<td>4.2.3</td>
<td>2.4.3</td>
<td>1.4.0</td>
<td>3.1.2</td>
<td>9.4.9</td>
<td>1.8.0</td>
<td>5.13.2</td>
<td>1.8.1</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`, `hiera`, and `mco` 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:

```plaintext
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 master 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`.

- MCollective plugins are installed in `/opt/puppetlabs/mcollective/plugins/` on *nix and in `<COMMON_APPDATA>\PuppetLabs\mcollective\etc\plugins\mcollective` on Windows. If you are adding new plugins to agent nodes, distribute them with Puppet.

**Related information**

Adding actions and plugins to PE on page 745

You can extend PE's MCollective engine by adding new actions. Actions are distributed in agent plugins, which are bundles of several related actions. You can write your own agent plugins (or download ones created by other people), and use PE to install and configure them on your nodes.

**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 master 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 app`.
  
  • **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.
  
  • **MCollective tools** — Tools used to invoke simultaneous actions across a number of nodes. These tools are built on the MCollective framework and are accessed via the `mco` command.
  
  • **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.
  
  • **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 these 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>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>RBAC data, including users, permissions, and AD/LDAP info.</td>
</tr>
<tr>
<td>pe-orchestrator</td>
<td>Orchestrator data, including details about job runs.</td>
</tr>
</tbody>
</table>

Use the native PostgreSQL tools to perform database exports and imports. At a minimum, you should 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-activemq</td>
<td>The ActiveMQ message server, which passes messages to the MCollective servers on agent nodes. Runs on servers with the master component.</td>
</tr>
<tr>
<td>pe-console-services</td>
<td>Manages and serves the console.</td>
</tr>
<tr>
<td>pe-puppetserver</td>
<td>Runs the master server, which manages the master component.</td>
</tr>
<tr>
<td>pe-nginx</td>
<td>Nginx, serves as a reverse-proxy to the console.</td>
</tr>
<tr>
<td>mcollective</td>
<td>The MCollective daemon, which listens for messages and invokes actions. Runs on every agent node.</td>
</tr>
</tbody>
</table>
### Service Definition

<table>
<thead>
<tr>
<th>Service</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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>
</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>peadmin</td>
<td>Invokes MCollective actions. The individual user account is the only user account intended for use in a login shell. This user exists on servers with the master component.</td>
</tr>
<tr>
<td>pe-puppet</td>
<td>Runs the master processes spawned by pe-puppetserver.</td>
</tr>
<tr>
<td>pe-webserver</td>
<td>Runs Nginx.</td>
</tr>
<tr>
<td>pe-activemq</td>
<td>Runs the ActiveMQ message bus used by MCollective.</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>
</tbody>
</table>

### Log files installed

The software distributed with PE generates log files that you can collect for compliance, or use for troubleshooting.

#### Master logs

The master has these logs.

- `/var/log/puppetlabs/puppetserver/puppetserver.log` — The master application logs its activity, including compilation errors and deprecation warnings, here.
- `/var/log/puppetlabs/puppetserver/puppetserver-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/puppetserver/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on compile masters.
- `/var/log/puppetlabs/puppetserver/code-manager-access.log`
- `/var/log/puppetlabs/puppetserver/file-sync-access.log`
- `/var/log/puppetlabs/puppetserver/masterhttp.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-access.log`
- `/var/log/puppetlabs/puppetserver/puppetserver.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-status.log`

© 2020 Puppet, Inc.
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.

ActiveMQ logs
ActiveMQ has these logs.
- /var/log/puppetlabs/activemq/wrapper.log
- /var/log/puppetlabs/activemq/activemq.log
- /var/log/puppetlabs/activemq/data/audit.log

MCollective logs
MCollective has these logs.
- /var/log/puppetlabs/mcollective.log — Maintained by the MCollective service, which is installed on all nodes.
- /var/log/puppetlabs/mcollective-audit.log — Exists on all nodes that have MCollective installed. Logs any MCollective actions run on the node, including information about the client that called the node

Console and console services logs
The console and pe-console-services has these logs.
- /var/log/puppetlabs/nginx/error.log — Contains errors related to nginx. Console errors that aren't logged elsewhere can be found in this log.
- /var/log/puppetlabs/nginx/access.log
- /var/log/puppetlabs/console-services/console-services.log
- /var/log/puppetlabs/console-services-access.log
- /var/log/puppetlabs/console-services/console-services-api-access.log
- /var/log/puppetlabs/console-services/console-services-daemon.log — This is where fatal errors or crash reports can be found.

Installer logs
The installer has these 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/installer-<timestamp>.log — Contains the operations performed and any errors that occurred during installation.
- /var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log — Contains the contents of the last installer run.
- /var/log/puppetlabs/installer/ <action_name>_<timestamp>_<run_description>.log — Contains details about high availability command execution. Each action triggers multiple Puppet runs, some on the master, some on the replica, so there might be multiple log files for each command.
Database logs

The database has these logs.

• /var/log/puppetlabs/puppetdb/puppetdb-access.log
• /var/log/puppetlabs/puppetdb/puppetdb-status.log
• /var/log/puppetlabs/puppetdb/puppetdb.log
• /var/log/puppetlabs/postgresql/pgstartup.log
• /var/log/puppetlabs/postgresql/postgresql-Fri.log
• /var/log/puppetlabs/postgresql/postgresql-Mon.log
• /var/log/puppetlabs/postgresql/postgresql-Sat.log
• /var/log/puppetlabs/postgresql/postgresql-Sun.log
• /var/log/puppetlabs/postgresql/postgresql-Thu.log
• /var/log/puppetlabs/postgresql/postgresql-Tue.log
• /var/log/puppetlabs/postgresql/postgresql-Wed.log

Orchestration logs

The orchestration service and related components have these logs.

• /var/log/puppetlabs/orchestration-services/orchestration-services.log
• /var/log/puppetlabs/orchestration-services/orchestration-services-access.log
• /var/log/puppetlabs/orchestration-services/orchestration-services-status.log
• /var/log/puppetlabs/orchestration-services/orchestration-services-daemon.log
— This is where fatal errors or crash reports can be found.
• /var/log/puppetlabs/orchestration-services/pcp-broker.log — The log file for Puppet Communications Protocol brokers on the master of masters.
• /var/log/puppetlabs/orchestration-services/pcp-broker-access.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.

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.

Services that run on the master or console — for example, pe-orchestration-services and pe-console-services — use the agent certificate to authenticate.

<table>
<thead>
<tr>
<th>Cert</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MASTER CERTNAME&gt;</td>
<td>Generated during install. In a monolithic install, this cert is used by PuppetDB and the console. This is the same value for the agent’s certname that runs on the master. In a monolithic install, the agent on the console and PuppetDB share this certname. In a default monolithic or split install, this is also the CA certificate.</td>
</tr>
<tr>
<td>&lt;CONSOLE CERTNAME&gt;</td>
<td>The certificate for the console, which is generated only if you have a split install. This is the same value for the agent’s certname that runs on the console.</td>
</tr>
<tr>
<td>&lt;PUPPETDB CERTNAME&gt;</td>
<td>The certificate for PuppetDB, which is generated only if you have a split install. This is the same value for the agent’s certname that runs on PuppetDB.</td>
</tr>
</tbody>
</table>
### Installing Puppet Enterprise

You can install PE in a monolithic configuration, where all infrastructure components are installed on one node, or in a split configuration, where the master, PuppetDB, and console are installed on separate nodes.

The installation method you use depends on the configuration you choose. For a monolithic configuration, you can use web-based installation or text mode, where you provide a configuration file to the installer. For a split configuration, you must install using text mode.

### Download and verify the installation package

PE is distributed in downloadable packages specific to supported operating system versions and architectures. Installation packages include the full installation tarball and a GPG signature (.asc) file used to verify authenticity.

#### Before you begin

You must have GnuPG installed.

1. **Download** the tarball appropriate to your operating system and architecture.
2. **Import** the Puppet public key.
   
   ```
   wget -O - https://downloads.puppetlabs.com/puppet-gpg-signing-key.pub | gpg --import
   ```
3. **Print** the fingerprint of the key.
   
   ```
   gpg --fingerprint 0x7F438280EF8D349F
   ```
   
   The primary key fingerprint displays: 6F6B 1550 9CF8 E59E 6E46 9F32 7F43 8280 EF8D 349F.
4. **Verify** the release signature of the installation package.
   
   ```
   $ gpg --verify puppet-enterprise-<version>-<platform>.tar.gz.asc
   ```
   
   The result is similar to:
   
   ```
   gpg: Signature made Tue 18 Sep 2016 10:05:25 AM PDT using RSA key ID EF8D349F
   gpg: Good signature from "Puppet, Inc. Release Key (Puppet, Inc. Release Key)"
   ```

**Note:** If you don't have a trusted path to one of the signatures on the release key, you receive a warning that a valid path to the key couldn't be found.

<table>
<thead>
<tr>
<th>Cert</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-internal-mcollective-servers</td>
<td>A certificate generated on the master and shared to all agent nodes.</td>
</tr>
<tr>
<td>pe-internal-peadmin-mcollective-client</td>
<td>The certificate for the peadmin account on the master.</td>
</tr>
<tr>
<td>pe-internal-puppet-console-mcollective-client</td>
<td>The MCollective certificate for the console.</td>
</tr>
</tbody>
</table>
Install using web-based installation (mono configuration)

Web-based installation uses a web server to guide you through installation. Use web-based installation for a monolithic configuration with or without compile masters.

Before you begin

Review the Web-based installation prerequisites on page 75.

1. Optional: If necessary, forward ports to the web-based installer.
   The web-based installer requires access to port 3000 on the machine you're running the installer from. If you can't connect directly to port 3000 — for example, if you're installing PE on a virtual machine or have firewall rules that prevent direct access — you can port forward, or "tunnel," to the installer using SSH.
   - *nix
     
     ssh -L 3000:localhost:3000 jumphost.example.tld
   
   - Windows
     a. Open PuTTY, select Sessions and in the Host Name field, enter the FQDN of the host you want to run the installer from.
     b. Select Tunnels and in the Source Port field, enter 3000.
     c. In the Destination field, enter localhost:3000.
     d. Select Local, click Add, and then click Open.

2. Unpack the installation tarball:
   
   tar -xf <TARBALL_FILENAME>

3. From the installer directory, run the installer:
   
   sudo ./puppet-enterprise-installer

4. When prompted, choose guided installation
   The installer starts a web server and provides an installer URL.
   
   The default installer URL is https://<INSTALL_PLATFORM_HOSTNAME>:3000. If you forwarded ports in step 1, the URL is https://localhost:3000.

5. In a browser, access the installer URL and accept the security request.
   The installer uses a default SSL certificate. You must add a security exception in order to access the installer.
   **Important:** Leave your terminal connection open until the installation is complete or else installation fails.

6. Follow the prompts to configure your installation.

7. On the validation page, verify the configuration and, if there aren't any outstanding issues, click Deploy now.
   Installation begins. You can monitor the installation as it runs by toggling Log View and Summary View. If you notice errors, check /var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log on the machine from which you're running the installer.
   When the installation completes, the installer script that was running in the terminal closes.

8. Click Start using Puppet Enterprise to log into the console.

Related information

Web-based installation prerequisites on page 75
Review these prerequisites and tips before beginning a web-based installation.

Web-based installation options on page 177
Use this reference when providing values in the web-based installer.

**Installing external PostgreSQL** on page 217

By default, Puppet Enterprise includes its own database backend, PE-PostgreSQL, which is installed alongside PuppetDB. If the load on your PuppetDB node is larger than it can effectively scale to (greater than 20,000 nodes), you can install a standalone instance of PE-PostgreSQL.

**Executable binaries and symlinks installed** on page 167

PE installs executable binaries and symlinks for interacting with tools and services.

### Install using text mode (mono configuration)

When you run the installer in text mode, you provide a configuration file (pe.conf) to the installer. The pe.conf file contains values for the parameters needed for installation.

1. Unpack the installation tarball:

   ```
   tar -xf <TARBALL_FILENAME>
   ```

2. From the installer directory, run the installer. The installation steps vary depending on the path you choose.

   - To use a pe.conf file that you've previously populated, run the installer **with the -c flag** pointed at the pe.conf file:
     ```
     sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
     ```
   - To have the installer open a copy of pe.conf for you to edit and install with, run the installer **without the -c flag**:
     ```
     sudo ./puppet-enterprise-installer
     ```
     a. Select **text-mode** when prompted.
     b. Specify required installation parameters.
     c. If you have an external server, refer to the external parameters in the pe.conf reference and add them to pe.conf.
     d. Save and close the file. Installation begins.

3. After the installation completes, run Puppet twice: `puppet agent -t`.

**Related information**

- **Text mode installer options** on page 178
- **Configuration parameters and the pe.conf file** on page 178
- **External PostgreSQL parameters** on page 181
- **Executable binaries and symlinks installed** on page 167

### Install using text mode (split configuration)

When you run the installer in text mode, you provide a configuration file (pe.conf) to the installer. The pe.conf file contains values for the parameters needed for installation.

Note: You must install the components in the order specified.
Install the master
Installing the master is the first step in setting up a split installation.

1. Unpack the installation tarball:

   ```bash
   tar -xf <TARBALL_FILENAME>
   ```

2. From the installer directory, run the installer. The installation steps vary depending on the path you choose.

   • To use a `pe.conf` file that you've previously populated, run the installer with the `-c` flag pointed at the `pe.conf` file:

   ```bash
   sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
   ```

   • To have the installer open a copy of `pe.conf` for you to edit and install with, run the installer without the `-c` flag:

   ```bash
   sudo ./puppet-enterprise-installer
   ```

   a. Select text-mode when prompted.
   b. Specify required installation parameters.
   c. If you have an external server, refer to the external parameters in the `pe.conf` reference and add them to `pe.conf`.
   d. Save and close the file. Installation begins.

3. When installation completes, transfer the installer and the `pe.conf` file located at `/etc/puppetlabs/enterprise/conf.d/` to the next server that you're installing a component on.

Install PuppetDB
In a split installation, after you install the master, you're ready to install PuppetDB.

1. Unpack the installation tarball:

   ```bash
   tar -xf <TARBALL_FILENAME>
   ```

2. From the installer directory, run the installer:

   ```bash
   sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
   ```

3. When installation completes, transfer the installer and the `pe.conf` file located at `/etc/puppetlabs/enterprise/conf.d/` to the next server that you're installing a component on.

Install the console
In a split installation, after you install the master and PuppetDB, you're ready to install the console.

1. Unpack the installation tarball:

   ```bash
   tar -xf <TARBALL_FILENAME>
   ```

2. From the installer directory, run the installer:

   ```bash
   sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
   ```

Run Puppet on infrastructure nodes
To complete a split installation, run Puppet on all infrastructure nodes in the order that they were installed.

1. Run Puppet on the master node.
2. Run Puppet on the PuppetDB node.
3. Run Puppet on the master node a second time.
4. Run Puppet on the console node.

**Related information**

- **Text mode installer options** on page 178
  When you run the installer in text mode, you can use the `--c` option to specify the full path to an existing `pe.conf` file. You can pair these additional options with the `--c` option.

- **Configuration parameters and the pe.conf file** on page 178
  A `pe.conf` file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

- **External PostgreSQL parameters** on page 181
  These parameters are required to install an external PostgreSQL instance. Password parameters can be added to standard installations if needed.

- **Executable binaries and symlinks installed** on page 167
  PE installs executable binaries and symlinks for interacting with tools and services.

**Web-based installation prerequisites**

Review these prerequisites and tips before beginning a web-based installation.

- If you've previously installed Puppet or Puppet Enterprise, make sure that the machine you're installing on is free of any artifacts left over from the previous installation.
- Make sure that DNS is properly configured on the machines you're installing on.
  - All nodes must know their own hostnames, which you can achieve by properly configuring reverse DNS on your local DNS server, or by setting the hostname explicitly. Setting the hostname usually involves the `hostname` command and one or more configuration files, but the exact method varies by platform.
  - All nodes must be able to reach each other by name, which you can achieve with a local DNS server, or by editing the `/etc/hosts` file on each node to point to the proper IP addresses.
- You can run the installer from a machine that is part of your deployment or from a machine that is outside your deployment. If you want to run the installer from a machine that is part of your deployment, in a split installation, run the installer from the node assigned the console component.
- The machine you run the installer from must have the same operating system and architecture as your deployment.
- The web-based installer does not support sudo configurations with `Defaults targetpw` or `Defaults rootpw`. Make sure your `/etc/sudoers` file does not contain, or comment out, those lines.
- For Debian users, if you gave the root account a password during installation of Debian, sudo may not have been installed. In this case, you must either install as root, or install sudo on any nodes on which you want to install.

**SSH prerequisites**

SSH requirements very depending on your installation method.

You don't need to take additional steps to configure SSH if you:

- Choose **Install on this server** during installation.
- Have a properly configured SSH agent with agent forwarding enabled.

If you're using SSH keys to authenticate across nodes, the public key for the user account performing the installation must be included in the `authorized_keys` file for that user account on each infrastructure node, including the machine from which you're running the installer. This requirement applies to root or non-root users.

<table>
<thead>
<tr>
<th>Installation method</th>
<th>Requirements</th>
<th>Prerequisites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root with a password</td>
<td>The installer requires the username and password for each infrastructure node.</td>
<td>Remote root SSH login must be enabled on each infrastructure node, including the node from which you're running the installer.</td>
</tr>
</tbody>
</table>
### Installation method

<table>
<thead>
<tr>
<th>Non-root with a password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root with an SSH key</td>
</tr>
</tbody>
</table>

### Requirements

- The installer requires the username, private key path, and key passphrase (as needed) for each infrastructure node.

### Prerequisites

- Sudo must be enabled for the non-root user on each infrastructure node.
- Remote root SSH login must be enabled on each node, including the node from which you're running the installer.
- The public root ssh key must be added to `authorized_keys` on each infrastructure node.
- The non-root user must be granted sudo access on each infrastructure node.
- The non-root user SSH key must be added to `authorized_keys` on each infrastructure node.

### Web-based installation options

Use this reference when providing values in the web-based installer.

SSH details are required only if you opt to install on another server.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Puppet master FQDN</strong></td>
<td>Fully qualified domain name of the server you're installing on. This FQDN is used as the name of the master certificate. This FQDN must be resolvable from the machine on which you're running the installer. To ensure you're using the proper FQDN for the monolithic master, run <code>sudo /opt/puppetlabs/bin/puppet config print certname</code> when the installation completes. If the installation fails because the FQDN value is incorrect, run the <code>config print certname</code> command and re-run the installer with the correct value.</td>
</tr>
<tr>
<td><strong>DNS altnames</strong></td>
<td>Comma-separated list of static, valid, DNS altnames so agents can trust the master. Make sure that this static list contains the DNS name or alias you're be configuring your agents to contact. The default settings include puppet.</td>
</tr>
<tr>
<td><strong>SSH username</strong></td>
<td>Username to use when connecting to the master. This user must either be root or have sudo access. The default value is root.</td>
</tr>
<tr>
<td><strong>SSH password</strong></td>
<td>Password associated with the SSH username. This password is used only if the user requires a password for sudo access.</td>
</tr>
</tbody>
</table>
### Setting | Value
--- | ---
SSH key file path | Absolute path to the SSH key on the machine you're performing the installation from. This value is used if an SSH password is not specified. Defaults to the root SSH key path.
SSH key passphrase | Passphrase for the SSH key, if applicable.

### Text mode installer options
When you run the installer in text mode, you can use the `-c` option to specify the full path to an existing `pe.conf` file. You can pair these additional options with the `-c` option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D</code></td>
<td>Display debugging information</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Run in quiet mode. The installation process isn't displayed. If errors occur during the installation, the command quits with an error message.</td>
</tr>
<tr>
<td><code>-y</code></td>
<td>Run automatically using the <code>pe.conf</code> file at <code>/etc/puppetlabs/enterprise/conf.d/</code>. If the file is not present or is invalid, installation or upgrade fails.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Display verbose debugging information.</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Display help information.</td>
</tr>
<tr>
<td><code>force</code></td>
<td>For upgrades only, bypass PostgreSQL migration validation. This option must appear last, after the end-of-options signifier (<code>--</code>), for example <code>sudo ./puppet-enterprise-installer -c pe.conf -- --force</code></td>
</tr>
</tbody>
</table>

### Configuration parameters and the `pe.conf` file
A `pe.conf` file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

You can create or obtain a `pe.conf` file by:

- Using the example `pe.conf` file provided in the `conf.d` directory in the installer tarball. This example file contains the mandatory parameters needed for a monolithic or split installation.

  **Tip:** In most cases, for a monolithic installation, you can use the example `pe.conf` file without making any changes.

- Selecting the text-mode installation option when prompted by the installer. This option opens your default text editor with the example `pe.conf` file, which you can modify as needed. Installation proceeds using that `pe.conf` after you quit the editor.

- Using the web-based installer to create a `pe.conf` file. After you run the web-based installer, you can find the file at `/etc/puppetlabs/enterprise/conf.d/`. You can also download the file by following the link provided on the confirmation page of the web-based installer.

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;master.example.com&quot;</td>
</tr>
<tr>
<td>Strings</td>
<td>&quot;console_admin_password&quot;: &quot;mypassword&quot;</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------</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 true or false (case sensitive, no quotation marks).</td>
</tr>
<tr>
<td></td>
<td>Note: Don't use Yes (y), No (n), 1, or 0.</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>

**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 master and provide a console administrator password after running the installer.

**puppet_enterprise::puppet_master_host**

The FQDN of the node hosting the master, for example `master.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.
   Default: pe-classifier-write

puppet_enterprise::classifier_database_super_user
   Classifier database superuser.
   Default: 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-puppetdb

© 2020 Puppet, Inc.
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"

**Master parameters**

Use these parameters to configure and tune the master.

**pe_install::puppet_master_dnsaltnames**

An array of strings that represent the DNS altnames to be added to the SSL certificate generated for the master.

Default: ["puppet"]

**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::code_manager_auto_configure**

true to automatically configure the Code Manager service, or false.

**puppet_enterprise::profile::master::r10k_remote**

String that represents 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 that’s supported by r10k and Git. This parameter is required only if you want r10k configured when PE is installed; it must be specified in conjunction with puppet_enterprise::profile::master::r10k_private_key.

**puppet_enterprise::profile::master::r10k_private_key**

String that represents the local file system path on the master where the SSH private key can be found and used by r10k, for example "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa". This
parameter is required only if you want r10k configured when PE is installed; it must be specified in conjunction with puppet_enterprise::profile::master::r10k_remote.

**puppet_enterprise::profile::master::check_for_updates**

true to check for updates whenever the pe-puppetserver service restarts, or false.

Default: true

### 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 master.

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 may 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 may be valid. May be set to "0" to not expire before the maximum token lifetime.

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::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**

Integer representing the number of days to wait before pruning the size of the classifier database. If you set the value to "0", the node classifier service is never pruned.

**puppet_enterprise::profile::console::classifier_node_check_in_storage**

"true" to store an explanation of how nodes match each group they're classified into, or "false".

Default: false
**puppet_enterprise::profile::console::display_local_time**

"true" to display timestamps in local time, with hover text showing UTC time, or "false" to show timestamps in UTC time.

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.

**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 master, across all orchestrator jobs.

Default: "8" requests

**puppet_enterprise::profile::orchestrator::job_prune_threshold**

Integer representing the days after which job reports should be removed.

Default: "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 tasks that can run at the same time.

Default: "250" tasks
puppet_enterprise::profile::orchestrator::use_application_services
true to enable application management, or false.
Default: false

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 475
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"}.

© 2020 Puppet, Inc.
**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"}.  

**Purchasing and installing a license key**

You can download and install Puppet Enterprise on 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, PE logs license warnings until you install an appropriate license key.

**Getting licensed**

When you have 11 or more active nodes, purchase a license key file.

- New PE customers — Purchase the license key file from the Puppet website or by contacting our sales team. Find contact information on the Ready to buy Puppet Enterprise? page.
- Existing PE customers — Add nodes by contacting your sales representative or emailing sales@puppet.com.

If you need more licenses, free up licenses by removing inactive nodes nodes from your deployment. By default, unused nodes are deactivated automatically after seven days with no activity (no new facts, catalog, or reports).

**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`.
   - For monolithic installations, add the file to the master node.
   - For split installations, add the file to the master and console nodes.
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
   ```

**Verify installed licenses and active nodes**

Check the number of active nodes in your deployment, the number of licenses, and the expiration date for your license.

Use the command line or console to view licensing details for your environment.

- On the master, run `puppet license`.
- In the console, click **License**.

**Tip:** The console caches license information, so changes to your license key file might not appear in the console for up to 24 hours. To refresh the cache immediately, manually restart pe-console-services.

**Installing agents**

You can install Puppet Enterprise agents on *nix, Windows, and macOS.

The master hosts a package repo used to install agents in your infrastructure.

The PE package management repo is created during installation of the master and serves packages over HTTPS using the same port as the master (8140). This means agents don't require any new ports to be open other than the one they already need to communicate with the master.

© 2020 Puppet, Inc.
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 OS on which it's running, sets up an apt, yum, or zipper repo that refers back to the master, 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 master.
- For *nix agents, and optionally Windows agents, downloads a tarball of plugins from the master. This feature is controlled by the settings pe_repo::enable_bulk_pluginsync and pe_repo::enable_windows_bulk_pluginsync, which you can configure in Hiera or in the console. For *nix agents, bulk plugin sync is set to true (enabled) by default. For Windows agents, the default is false (disabled).
- 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 master 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>
<tbody>
<tr>
<td>puppet.conf settings</td>
<td>agent:splay=true</td>
<td>The puppet.conf file looks like this:</td>
</tr>
<tr>
<td></td>
<td>agent:certname=node1.corp.net</td>
<td></td>
</tr>
<tr>
<td></td>
<td>agent:environment=development</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[agent]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>certname = node1.corp.net</td>
<td></td>
</tr>
<tr>
<td></td>
<td>splay = true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>environment = development</td>
<td></td>
</tr>
</tbody>
</table>

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:

```yaml
---
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.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 master, 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
curl -k https://master.example.com:8140/packages/current/install.bash | sudo bash -s agent:certname=<certnameOtherThanFQDN>
custom_attributes:challengePassword=<passwordForAutosignerScript>
extension_requests:pp_role=<puppetNodeRole>
```

* Windows install script with example agent setup and certificate signing parameters:

```powershell
\install.ps1 agent:certname=<certnameOtherThanFQDN>
```
custom_attributes:challengePassword=<passwordForAutosignerScript>
extension_requests:pp_role=<puppetNodeRole>

**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>
<tr>
<td>PUPPET_AGENT_ACCOUNT_USER</td>
<td>-PuppetAgentAccountUser</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_PASSWORD</td>
<td>-PuppetAgentAccountPassword</td>
</tr>
<tr>
<td>PUPPET_AGENT_ACCOUNT_DOMAIN</td>
<td>-PuppetAgentAccountDomain</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  • stopped</td>
</tr>
<tr>
<td>enable</td>
<td>--puppet-service-enable &lt;VALUE&gt;</td>
<td>-PuppetServiceEnable &lt;VALUE&gt;</td>
<td>• true       • false       • manual (Windows only) • mask</td>
</tr>
</tbody>
</table>

For example:

*nix

```bash
curl -k https://master.example.com:8140/packages/current/install.bash | sudo bash -s -- --puppet-service-ensure stopped
```

**Windows**

```powershell
```

© 2020 Puppet, Inc.
Installing *nix agents

PE has package management tools to help you easily install and configure agents. You can also use standard *nix package management tools.

Install *nix agents with PE package management

PE provides its own package management to help you install agents in your infrastructure.

Note: The <MASTER HOSTNAME> portion of the installer script—as provided in the following example—refers to the FQDN of the master. 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 master, add the appropriate class for the repo that contains the agent packages.
   a) In the console, click Classification, and in the PE Infrastructure group, select the PE Master group.
   b) On the Configuration tab in the Class name field, 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 master node 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

         curl -k https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash

      Tip: On 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.
   • wget

         wget -O - -q --no-check-certificate --secure-protocol=TLSv1 https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash

   • 10 (run as root)

         export PATH=$PATH:/opt/sfw/bin
         wget -O - -q --no-check-certificate --secure-protocol=TLSv1 https://<MASTER HOSTNAME>:8140/packages/current/install.bash | bash

3. Sign the agent's certificate.

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.

Agent packages can be found on the master 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 master 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.
If your nodes are running an operating system or architecture that is different from the master, download the appropriate agent package, extract the agent packages into the appropriate repo, and then install the agents on your nodes just as you would any other package, for example `yum install puppet-agent`.

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
     ```
4. Configure the agent as needed: `puppet config set`

### Install *nix agents using a manually transferred certificate

If you choose not to or can't use `curl -k` to trust the master during agent installation, you can manually transfer the master CA certificate to any machines you want to install agents on, and then run the installation script against that cert.

**Note:** `-k` is not supported for AIX 5.3, 6.1, and 7.1. You must replace the `-k` flag with `-tlsv1` or `-l`.

1. On the machine that you're installing the agent on, create the directory `/etc/puppetlabs/puppet/ssl/certs/`.
2. On the master, 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:
   ```
   curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash
   ```
5. Sign the agent's certificate.

### 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.

#### Before you begin

Download the appropriate agent tarball.

### 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.

**Note:** You must repeat this process each time you upgrade your master.

1. On your master, 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-1.10.4`.
2. Run Puppet: `puppet agent -t`
3. Follow the steps for Install *nix agents with PE package management on page 190.

### 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.

**Note:** You must repeat this process each time you upgrade your master.
1. Add the agent package to your own package management and distribution system.
2. Disable the PE-hosted repo.
   a) In the console, click Classification, and in the PE Infrastructure group, select the PE Master group.
   b) On the Configuration 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 compile masters using your own package management without internet access**
If your infrastructure relies on compile masters to install agents, you don’t have to copy the agent package to each compile master. Instead, use the console to specify a path to the agent package on your package management server.

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 Classification, and in the PE Infrastructure group, select the PE Master group.
   b) On the Configuration tab, find the pe_repo class and specify the parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_path</td>
<td>FQDN of your package management server.</td>
</tr>
</tbody>
</table>

c) Click Add parameter and commit changes.

**Installing Windows agents**
You can install Windows agents with PE package management, with a manually transferred certificate, or with the Windows .msi package.

**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 master doesn't have internet access, download the appropriate agent package and save it on your master 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 <MASTER_HOSTNAME> portion of the installer script—as provided in the following example—refers to the FQDN of the master. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. In the console, click Classification, and in the PE Infrastructure group, select the PE Master group.
2. On the Configuration tab in the Class name field, select 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 master, run Puppet to configure the newly assigned class.

The new repository is created on the master at /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>/.
5. On the node, open an administrative PowerShell window, and install:

```powershell
```

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 master 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 master to the node you're installing.

<table>
<thead>
<tr>
<th>File</th>
<th>Location on master</th>
<th>Location on target node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation script (install.ps1)</td>
<td>/opt/puppetlabs/server/data/packages/public/</td>
<td>Any accessible local directory.</td>
</tr>
<tr>
<td>CA certificate (ca.pem)</td>
<td>/etc/puppetlabs/puppet/ssl/certs/</td>
<td>C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\</td>
</tr>
</tbody>
</table>

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 master.

1. Run the installer as administrator.
2. When prompted, provide the hostname of your master, for example puppet.
3. Sign the agent's certificate.
**Install Windows agents using msiexec from the command line**

Install the MSI manually from the the command line if you need to customize puppet.conf, CSR attributes, or certain agent properties.

On the command line of the node that you want to install the agent on, run the install command:

```
msiexec /qn /norestart /i puppet.msi
```

**Tip:** You can specify `/l*v install.txt` to log the progress of the installation to a file.

**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_MASTER_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 used these properties once, 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 pe.conf</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_MASTER_SERVER</td>
<td>Hostname where the master can be reached.</td>
<td>server</td>
<td>puppet</td>
</tr>
<tr>
<td>PUPPET_CA_SERVER</td>
<td>Hostname where the CA master can be reached, if you're using multiple masters and only one of them is acting as the CA.</td>
<td>ca_server</td>
<td>Value of PUPPET_MASTER_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.
### PUPPET_AGENT_STARTUP_MODE

Whether and how the agent service is allowed to run. Allowed values are:

- **Automatic** — Agent starts up when Windows starts and remains running in the background.
- **Manual** — Agent can be started in the services console or with `net start` on the command line.
- **Disabled** — Agent is installed but disabled. You must change its startup type in the services console before you can start the service.

**Setting in `pe.conf`**: n/a

**Default**: Automatic

### PUPPET_AGENT_ACCOUNT_USER

Windows user account the agent service uses. 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 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`.

**Setting in `pe.conf`**: n/a

**Default**: LocalSystem

### PUPPET_AGENT_ACCOUNT_PASSWORD

Password for the agent’s user account.

**Setting in `pe.conf`**: n/a

**Default**: No Value
### Property Definitions

<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_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. <strong>Important:</strong> If you need to downgrade agents, use <code>REINSTALLMODE=amus</code> when calling <code>msiexec.exe</code> at the command line to prevent removing files that the application needs.</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>

To install the agent with the master at `puppet.acme.com`:

```
msiexec /qn /norestart /i puppet.msi PUPPET_MASTER_SERVER=puppet.acme.com
```

To install the agent to a domain user `ExampleCorp\bob`:

```
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 master 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. After the first Puppet run, the MCollective service starts and the node can be controlled with MCollective. The agent service and the MCollective service can be started and stopped independently using either the service control manager `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, Facter, and MCollective. 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>mcollective</td>
<td>MCollective 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.

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.

**Installing 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 190.

**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 lowercase 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 master hostname and the agent certname.
   - Tip: You can get the master hostname with `puppet cert list --all`.
3. Sign the agent’s certificate.
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 master:
   `/opt/puppetlabs/bin/puppet config set server <MASTER_HOSTNAME>`
7. Configure the agent certname:
   `/opt/puppetlabs/bin/puppet config set certname <AGENT_CERTNAME>`
8. Sign the agent’s certificate.

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.

Installing 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 monolithic master. The root user who performs this installation can then set up non-root users on the master 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

Before you begin
You must have a monolithic installation, and because non-root users can’t use capabilities to manage nodes, must be disabled. (In the console, in the PE MCollective group, on the Rules tab, remove the `aio_agent_version` fact.)

Note: Unless specified otherwise, perform these steps as a root user.

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:

```
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:

```
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 service:

```
puppet resource service mcollective ensure=stopped enable=false
```


   a) In the console, click Classification, and in the PE Infrastructure group, select the PE Agent group.
   
   b) On the Configuration tab, select the puppetEnterprise::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>

7. Change to the non-root user and generate a certificate signing request:

```
puppet agent -t --certname "<UNIQUE NON-ROOT USERNAME.HOSTNAME>" --server "<MASTER 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.

8. On the master or in the console, approve the certificate signing request.

9. On the agent node as the non-root user, set the node’s certname and the master hostname, and then run Puppet.

```
puppet config set certname <UNIQUE NON-ROOT USERNAME.HOSTNAME> --section agent

puppet config set server <PUPPET MASTER HOSTNAME> --section agent

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 master when a non-root user runs Puppet:
  ```
puppet agent -t
  ```

- The agent service is not running: `service puppet status`

- The non-root user isn’t listed in the console in the PE MCollective group.

© 2020 Puppet, Inc.
• 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**

**Before you begin**

You must have a monolithic installation, and because non-root users can’t use capabilities to manage nodes, must be disabled. (In the console, in the PE MCollective group, on the Rules tab, remove the `aio_agent_version` fact.)

**Note:** Unless specified otherwise, perform these steps as a root user.

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
   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
   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. Disable the service:

   ```bash
   puppet resource service mcollective ensure=stopped enable=false
   
5. Change to the non-root user and generate a certificate signing request:

   ```bash
   puppet agent -t --certname "<UNIQUE NON-ADMIN USERNAME>" --server "<MASTER HOSTNAME>"
   
   **Important:** This Puppet run submits a cert request to the master 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.

6. As the non-root user, create a configuration file at `%USERPROFILE%/.puppet/puppet.conf` to specify the agent certname and the hostname of the master:

   ```
   [main]
   certname = <UNIQUE NON-ADMIN USERNAME>.hostname
   server = <MASTER HOSTNAME>
   ```

7. As the non-root user, submit a cert request: `puppet agent -t`.
8. On the master 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.


   The configuration specified in the catalog is applied to the node.
Non-root user functionality
Non-root users can only 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
- sshAuthorized_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
- user
- group
- service

Managing certificate signing requests
When you install a new PE agent, the agent automatically submits a certificate signing request (CSR) to the master.

Certificate requests can be signed from the console or the command line. If DNS altnames are set up for agent nodes, you must use the command line interface to approve and reject node requests.

After approving a node request, the node doesn’t show up in the console until the next Puppet run, which can take up to 30 minutes. You can manually trigger a Puppet run if you want the node to appear immediately.

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.
Managing certificate signing requests in the console
The console displays a list of nodes on the Unsigned certs page that have submitted CSRs. You can approve or deny CSRs individually or in a batch.

If you use the Accept All or Reject All options, processing could take up to two seconds per request.

When using 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.

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 puppet cert list

To sign a pending request:

$ sudo puppet cert sign <name>

To sign pending requests for nodes with DNS altnames:

$ sudo puppet cert sign (<HOSTNAME> or --all) --allow-dns-alt-names'

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 master called master.example.com, run puppet config set server master.example.com. This command adds the setting server = puppetmaster.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 network device agents
Install agents on network switches to operate them with Puppet Enterprise as managed devices.

Installing Arista EOS agents
You can run agents on Arista EOS network switches.

Install Arista EOS agents

1. On your master, install the netdev_stdlib_eos module: puppet module install aristanetworks-netdev_stdlib_eos

   This module contains the types and providers needed to run the Puppet agent on the network switch.
2. Install the agent on the network device.
   a) Access your network device as an admin user, or as a user with access to Privileged EXEC mode.
   b) Enable Privileged EXEC mode: enable
   c) On the EOS download page, locate the most recent .swix package for the agent and copy it:
      copy http://downloads.puppetlabs.com/eos/4/PC1/i386/puppet-agent-<VERSION NUMBER>.eos4.i386.swix extension:
      Note: If you’re unable to access the Internet from your EOS instance, download the agent package and
      transfer it to your instance.
   d) Install the agent on the network device: extension puppet-agent-<VERSION NUMBER>-eos-4-i386.swix
   e) Log out and log back into the network device as root.
   f) Set Puppet to run as the root user and group:
      /opt/puppetlabs/bin/puppet config --confdir /persist/sys/etc/puppetlabs/
      puppet set user root
      /opt/puppetlabs/bin/puppet config --confdir /persist/sys/etc/puppetlabs/
      puppet set group root
   g) Configure the agent to connect to your master: puppet config set server <PUPPET MASTER FQDN>
   h) Connect the agent to the master and create a certificate signing request for the new agent: puppet agent
      --test
3. On your master, sign the cert for the network device: puppet cert sign <NETWORK DEVICE FQDN>
4. On the network device, run Puppet: puppet agent -t
   The agent retrieves its catalog and is fully functional.

   You see a message like:

   Info: Retrieving pluginfacts
   Info: Retrieving plugin
   Info: Loading facts
   Info: Caching catalog for <EOS INSTANCE FQDN>
   Info: Applying configuration version '1424214157'
   Notice: Finished catalog run in 0.46 seconds

Uninstall Arista EOS agents

Note: If you’re uninstalling and reinstalling the agent for testing purposes, you must follow these instructions
completely to ensure you don't get SSL collisions when reinstalling.
1. Access your network device instance as an admin user.
2. Enable Privileged EXEC mode: enable
3. Delete the network device agent extension:
   no extension puppet-enterprise-<VERSION NUMBER>-eos-4-1386.swix
   delete extension:puppet-enterprise-<VERSION NUMBER>-eos-4-1386.swix
4. Delete the SSL keys from the EOS instance: bash sudo rm -rf /persist/sys/etc/puppetlabs/
5. On your master, revoke the cert for the agent on the network device instance: puppet cert clean <EOS
   INSTANCE FQDN>

   The agent certificate is revoked and and related files are deleted from the master.

   You see a message like:

   Notice: Revoked certificate with serial 10

© 2020 Puppet, Inc.
Installing Cisco agents

You can install agents on network switches running Cisco NX-OS 7.0 and later.

The ciscopuppet module allows you to manage Cisco Nexus Network Elements using Puppet. For a full list of supported platforms and limitations, see the Resource Platform Support Matrix.

The Cisco IOS module allows you to configure Cisco Catalyst devices running IOS.

Install Cisco NX-OS agents

The puppet-ciscopuppet module contains the types and providers needed to configure Cisco NX-OS network switches.

Before you begin

If you're installing for bash, you must have networking and DNS configured for the bash environment.

Important: You cannot install the bash environment agent on NX-OS 9.2(1) or newer.

NX-OS supports two possible environments for running third-party software. You can run Puppet from either environment but not both at the same time.

- Bash — Native Linux environment underlying NX-OS. It is disabled by default.
- Guest — Secure Linux container environment running CentOS. It is enabled by default on most Nexus platforms

1. On your master, install the puppetlabs-ciscopuppet module: puppet module install puppetlabs-ciscopuppet

2. From the network device, start a root bash or guest shell. For a bash install, enable networking:

   run bash
   sudo ip netns exec management bash

3. Install the Puppet Collections PC1 repository package:

   - Bash
     
     bash-4.2# yum install http://yum.puppetlabs.com/puppetlabs-release-pcl-cisco-wrlinux-5.noarch.rpm
   
   - Guest
     
     sudo su -
     yum install http://yum.puppetlabs.com/puppetlabs-release-pcl-el-7.noarch.rpm

4. Install the agent: yum install puppet-agent

   The agent is installed in /opt/puppetlabs.

5. Add the bin and lib paths to your login shell's PATH: export PATH=$PATH:/opt/puppetlabs/puppet/bin:/opt/puppetlabs/puppet/lib

Uninstall Cisco agents

The commands to uninstall NX-OS Cisco agents are the same for both Bash and guest shell environments.
1. Remove the agent:

   ```bash
   sudo su -
   yum remove puppet-agent
   ```

2. Remove the Puppet Collections PC1 repository package package:

   ```bash
   yum remove puppetlabs-release-pc1
   ```

## Installing compile masters

As your Puppet Enterprise infrastructure scales up to 4,000 nodes and beyond, add load-balanced compile masters to your monolithic installation to increase the number of agents you can manage.

Each compile master increases capacity by 1,500 to 3,000 nodes, until you exhaust the capacity of PuppetDB or the console.

### How compile masters work

A single master can process requests and compile code for up to 4,000 nodes. When you exceed this scale, expand your infrastructure by adding compile masters to share the workload and compile catalogs faster.

**Important:** Compile masters must run the same OS major version, platform, and architecture as the MoM.

Compile masters act as PCP brokers, conveying messages between the orchestrator and Puppet Execution Protocol (PXP) agents. PXP agents connect to PCP brokers running on compile masters over port 8142. Status checks on compile masters must be sent to port 8140, using `https://<hostname>:8140/status/v1/simple`.

### Components and services running on compile masters

All compile masters contain a Puppet Server and a file sync client.

When triggered by a web endpoint, file sync takes changes from the working directory on the MoM and deploys the code to a live code directory. File sync then deploys that code to all your compile masters, ensuring that all masters in a multi-master configuration remain in sync. By default, compile masters check for code updates every five seconds.

The certificate authority (CA) service is disabled on compile masters. A proxy service running on the compile master Puppet Server directs CA requests to the MoM, which hosts the CA in default installations.

Compile masters also have:

- The repository for agent installation, `pe_repo`
- The controller profile used with PE client tools

© 2020 Puppet, Inc.
• Puppet Communications Protocol (PCP) brokers to enable orchestrator scale

Logs for compile masters are located at /var/log/puppetlabs/puppetserver/.

Logs for PCP brokers on compile masters are located at /var/log/puppetlabs/puppetserver/pcp-broker.log.

Using load balancers with compile masters

When using more than one compile master, a load balancer can help distribute the load between the compile masters 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 in the HAproxy module documentation.

Load balancing

PCP brokers run on compile masters 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 will isolate themselves if they become disconnected. You can check connections with the /status/v1/simple endpoint for an error state.

You must also 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 15m since PCP brokers disconnect inactive connections after 15 minutes.

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 master 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 master, 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.

Optimizing workload distribution

Due to the diverse nature of the network communications between the agent and the master, we recommend that you implement a load balancing algorithm that distributes traffic between compile masters based on the number of open connections. Load balancers often refer to this strategy as “balancing by least connections.”

Related information

Firewall configuration for monolithic installations with compile masters on page 160

These are the port requirements for monolithic installations with compile masters.

Firewall configuration for large environment installations on page 163

The port requirements for large environment installation are the same as those for split installation.

GET /status/v1/simple on page 353
The /status/v1/simple returns a status that reflects all services the status service knows about.

**Install compile masters**

To install a compile master, you first install an agent and then classify that agent as a compile master.

1. SSH into the node that you want to make a compile master and install the agent:

   ```
curl -k https://<MoM_HOSTNAME>:8140/packages/current/install.bash | sudo bash -s main:dns_alt_names=<COMMA-SEPARATED LIST OF ALT NAMES FOR THE COMPILE MASTER>
   ```

   **Note:** Set the `dns_alt_names` value to a comma-separated list of any alternative names that agents use to connect to compile masters. The installation uses puppet by default. If your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter.

2. From the master of masters, sign the compile master's certificate:

   ```
puppet cert --allow-dns-alt-names sign <COMPILE_MASTER_HOSTNAME>
   ```

   **Note:** You can't use the console to sign certs for nodes with DNS alt names.

3. Pin the compile master node to the **PE Master** node group.

   a) In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) Enter the **Certname** for the compile master, click **Pin node**, and then commit changes.

4. From the compile master, run Puppet: `puppet agent -t`

5. From the master of masters, run Puppet: `puppet agent -t`

After installing compile masters, you must configure them to appropriately route communication between your master of masters and agent nodes.

**Configure compile masters**

Compile masters must be configured to appropriately route communication between your master of masters and agent nodes.

**Before you begin**

- Install compile masters and load balancers.
- If you need DNS altnames for your load balancers, add them to the master.
- Ensure port 8143 is open on the master of masters or on any workstations used to run orchestrator jobs.

1. Configure `pe_repo::compile_master_pool_address` to send agent install requests to the load balancer.

   **Important:** If you have load balancers in multiple data centers, you must configure `compile_master_pool_address` using Hiera, instead of using configuration data in the console, as described in this step. Using either of these methods updates the agent install script URL displayed in the console.

   a) In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) Select the **Configuration** tab, and in the **Data** section, in the **pe_repo** class, 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

   - Compile master
   - PuppetDB node (split configurations only)
   - Console node (split configurations only)
   - Master of masters
3. Configure infrastructure agents to connect orchestration agents to the master of masters.
   a) In the console, click Classification, and in the PE Infrastructure group, select the PE Infrastructure Agent group.
   b) If you manage your load balancers with agents, on the Rules tab, pin load balancers to the group.
   c) On the Configuration tab, find the puppet_enterprise::profile::agent class and specify parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
| pcp_broker_list | JSON list including the hostname for your master of masters. If you have an HA replica, include it after the MoM. Hostnames must include port 8142, for example 
"MASTER.EXAMPLE.COM:8142". |
| master_uris     | Provide the host name for your master of masters, for example, ["https://MOM.EXAMPLE.COM"]. Uris must begin with https://. This setting assumes port 8140 unless you specify otherwise with host:port. |

   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.

   This Puppet run doesn't change PXP agent configuration. If you have high availability configured and haven’t already pinned your load balancer to the PE Infrastructure Agent group, the Puppet run configures your load balancer to compile catalogs on the MoM.

4. Configure agents to connect orchestration agents to the load balancer.
   a) In the console, click Classification, and in the PE Infrastructure group, select the PE Agent group.
   b) On the Configuration tab, find the puppet_enterprise::profile::agent class and specify parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
| pcp_broker_list | JSON list including the hostname for your load balancers. Hostnames must include port 8142, for example 
"LOADBALANCER1.EXAMPLE.COM:8142", "LOADBALANCER2.EXAMPLE.COM". |
| master_uris     | Provide a list of load balancer host names, for example, ["https://LOADBALANCER1.EXAMPLE.COM","https://LOADBALANCER2.EXAMPLE.COM"]. Uris must begin with https://. This setting assumes port 8140 unless you specify otherwise with host:port. |

   c) Remove any values set for pcp_broker_ws_uris.
   d) Commit changes.
   e) Run Puppet on the master, 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 monolithic installations with compile masters on page 160
These are the port requirements for monolithic installations with compile masters.

Firewall configuration for large environment installations on page 163
The port requirements for large environment installation are the same as those for split installation.

## Installing ActiveMQ hubs and spokes

Add hubs and spokes to large Puppet Enterprise deployments for efficient load balancing and for relaying MCollective messages.

Adding hubs and spokes can be done in addition to, or independently from, adding additional masters to your infrastructure.

### Setting up MCollective

Because MCollective does not install with PE you must enable it.

**CAUTION:**

2018.1 is the last release to support Marionette Collective, also known as MCollective. While 2018.1 remains supported, will continue to address security issues for MCollective. Feature development has been discontinued. Future releases of will not include MCollective. For more information, see the [Puppet Enterprise support lifecycle](#).

To prepare for these changes, migrate your MCollective work to orchestrator to automate tasks and create consistent, repeatable administrative processes. Use orchestrator to automate your workflows and take advantage of its integration with console and commands, APIs, role-based access control, and event tracking.

### Procedure

- Before you install PE 2018.1 on the master, add the following parameter to your `pe.conf` file:

  
  "pe_install::disable_mco": false

### Related information

- [Configuration parameters and the `pe.conf` file](#) on page 178
- A `pe.conf` file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

- [Move from MCollective to Puppet orchestrator](#) on page 751
- Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

## Install ActiveMQ hubs and spokes

Setting up hubs and spokes involves classifying nodes into appropriate node groups, and then configuring the connections for those node groups.

### Before you begin

You must have a monolithic or split deployment with all infrastructure servers running the same OS and architecture.
Install the agent on ActiveMQ hub and spoke nodes
Hub and spoke nodes must have an agent installed.

1. SSH into each machine that you want to operate as a hub or spoke and install Puppet.

```
curl -k https://<MASTER.EXAMPLE.COM>:8140/packages/current/install.bash | sudo bash -s agent:ca_server=<MASTER.EXAMPLE.COM>
```

2. Sign the nodes' certificates.

**Create the ActiveMQ hub group**
Create the ActiveMQ hub group and pin the hub node to the group.

1. In the console, click **Classification**, and then click **Add group**.
2. Specify options for the new node group and then click **Add**.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent name</td>
<td>PE Infrastructure</td>
</tr>
<tr>
<td>Group name</td>
<td>PE ActiveMQ Hub</td>
</tr>
<tr>
<td>Environment</td>
<td>Select the environment agents are in.</td>
</tr>
<tr>
<td>Environment group</td>
<td>Do not select this option.</td>
</tr>
</tbody>
</table>

3. Click the link to **Add membership rules, classes, and variables**.
4. On the **Rules** tab in the **Node name** field, enter the hostname for the hub and click **Pin node**.
5. On the **Configuration** tab in the **Class name** field, enter `puppetEnterprise::profile::amq::hub`, and click **Add class**.
6. In the `puppetEnterprise::profile::amq::hub` class, specify parameters and then commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_connector_spoke_collect_tag</td>
<td>pe-amq-network-connectors-for-&lt;HUB_HOSTNAME&gt;</td>
</tr>
</tbody>
</table>

7. Run Puppet on the hub node.
Add spokes to ActiveMQ broker group
Add spoke nodes to the **PE ActiveMQ Broker** group, which is a preconfigured node group.

1. In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE ActiveMQ Broker** group.
2. On the **Rules** tab in the **Node name** field, enter the hostname for each spoke and click **Pin node**.
3. On the **Configuration** tab in the *puppet_enterprise::profile::amq::broker* class, specify parameters and then commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activemq_hubname</td>
<td>Hub FQDN, entered as an array, for example &quot;[&quot;ACTIVEMQ-HUB.EXAMPLE.COM&quot;].&quot;</td>
</tr>
</tbody>
</table>

4. Run Puppet on all spoke nodes, and then run Puppet on the hub node.

Create a custom fact for node and spoke relationships
In a hub and spoke configuration, all the nodes other than infrastructure nodes use the most suitable spoke as their broker. You create these connections with custom facts.

Suitable spokes are usually those that share a geographic location or share network segments. In some circumstance the spokes may also be behind a load balancer.

Create a custom fact.
For example, create a custom fact that represents a Sydney data center.

- *nix

  ```
  puppet apply -e 'file { 
  ["/etc/puppetlabs", "/etc/puppetlabs/facter", "/etc/puppetlabs/facter/facts.d"]:
  ensure => directory }'
  puppet apply -e 'file {"/etc/puppetlabs/facter/facts.d/data_center.txt":
  ensure => file, content => "data_center=syd"}
  ```

- Windows

  ```
  puppet apply -e "file { 
  ['C:/ProgramData/PuppetLabs', 'C:/ProgramData/PuppetLabs/facter', 
  'C:/ProgramData/PuppetLabs/facter/facts.d']:
  ensure => directory }"
  puppet apply -e "file {'C:/ProgramData/PuppetLabs/facter/facts.d/
  data_center.txt': ensure => file, content => 'data_center=syd'}"
  ```

Classify the ActiveMQ spokes
Use custom facts to classify spokes in the console or to bind agents to spokes with Hiera.

Classify the ActiveMQ spokes with the console
Use the console to create new node groups for each spoke or group of spokes in your infrastructure.

Groups must belong to the **PE MCollective** group, and include the *puppet_enterprise::profile::mcollective::agent* class, with the *activemq_brokers* parameter set to the name of the desired spokes.

1. In the console, click **Classification**, and then click **Add group**.
2. Specify options for the new node group, and then click **Add**.

For example, create a group that represents a Sydney datacenter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent name</strong></td>
<td>PE MCollective</td>
</tr>
<tr>
<td><strong>Group name</strong></td>
<td>Sydney_datacenter</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>Select the environment agents are in.</td>
</tr>
<tr>
<td><strong>Environment group</strong></td>
<td>Do not select this option.</td>
</tr>
</tbody>
</table>

3. Click the link to **Add membership rules, classes, and variables**.

4. On the **Rules** tab, create a rule to add agents to this group, then click **Add rule** and commit changes.

For example, create a rule that matches nodes in the Sydney datacenter.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fact</strong></td>
<td>data_center</td>
</tr>
<tr>
<td><strong>Operator</strong></td>
<td>=</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>syd</td>
</tr>
</tbody>
</table>

5. On the **Configuration** tab in the **Add new class** field, enter `puppetenterprise::profile::mcollective::agent` and click **Add class**.

6. In the `puppetenterprise::profile::mcollective agent` class, specify parameters and then commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>activemq_brokers</strong></td>
<td>Names of the spokes you want to classify. Hubs must be entered as an array.</td>
</tr>
</tbody>
</table>

7. Run on the hub and spokes, including the master or master of masters, and on any agents.

**Classify the ActiveMQ spokes with Hiera**

If you do not want to use the console to create new node groups for each spoke or groups of spokes in your infrastructure, you can use Hiera with automatic data binding instead.

You must remove the `mcollective_middleware_hosts` parameter from the `puppetenterprise` class in the **PE Infrastructure** group, and place this parameter within Hiera at the appropriate level to distinguish the different spokes.

1. On the master, edit your Hiera config file (`/etc/puppetlabs/puppet/hiera.yaml`) so that it contains, as part of the hierarchy, the custom fact you're using to classify spokes.

For example, add the custom **data_center** fact to the hierarchy.

```yaml
#hiera.yaml
---
  :backends:
    - eyaml
    - yaml
  :hierarchy:
    - "%{clientcert}"  
    - "%{data_center}"  
    - global
  :yaml:
    :datadir: "/etc/puppetlabs/code/environments/%{environment}/hieradata"
```

© 2020 Puppet, Inc.
2. On the master, add Hiera data files to map ActiveMQ spokes to custom facts.
   
   For example, map spokes to Sydney and Portland datacenters using the custom data_center fact.
   
   a. Navigate to /etc/puppetlabs/code/environments/production/hieradata/, and create a file called syd.yaml that contains the content:

   ```yaml
   puppet_enterprise::profile::mcollective::agent::activemq_brokers:
     - 'SPOKE.SYD.EXAMPLE.COM'
   ```

   b. Still in the hieradata directory, create a file called pdx.yaml that contains the content:

   ```yaml
   puppet_enterprise::profile::mcollective::agent::activemq_brokers:
     - 'SPOKE.PDX.EXAMPLE.COM'
   ```

3. Verify the custom fact on the end node.
   
   For example, running facter data_center on a node classified in the Sydney datacenter returns the value syd.

4. Verify that Hiera picks up the expected value for the ActiveMQ spoke given the appropriate parameters.
   
   For example, on the master, running this command returns the value of the Sydney broker hostname.

   ```bash
   hiera puppet_enterprise::profile::mcollective::agent::activemq_brokers data_center=syd environment=production
   ```

5. On the master, reload the pe-puppetserver service: `sudo service pe-puppetserver reload`.

6. Run on the hub and spokes, including the master or master of masters, and on any agents.

Verify connections in your infrastructure

Check that ActiveMQ hub and spokes are configured correctly.

1. Verify that the MCollective group is correctly configured by accessing the master hostname and running `su puppetadmin` and then `mco ping`.

   The hub and spokes, including the master and any agents, are listed.

2. Verify that the hub's connections are correctly established. Access the ActiveMQ hub hostname and run:

   - RHEL 7 and derivatives — `ss -a -n | grep '61616'`
   - Other platforms — `netstat -an | grep '61616'`

   The hub displays connections to ActiveMQ broker nodes.

### Installing PE client tools

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily 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 master.

Client tools versions align with PE versions. For example, if you're running PE 2017.3, you should use the 2017.3 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.

**Important:** When you upgrade PE to a new "x.y" version, install the appropriate "x.y" version of 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-app`.
• 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 master 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 480

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 Puppet master or from a separate work station, there are two types of configuration files: a global configuration file and a user-specified configuration file.

**Configuring puppet-access** on page 290

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 Puppet master.

**Installing and configuring puppet-code** on page 587

PE automatically installs and configures the `puppet-code` command on your masters 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 these platforms.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Versions</th>
<th>Arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>CentOS</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>Oracle Linux</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>Scientific Linux</td>
<td>6, 7</td>
<td>x86_64</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>11, 12</td>
<td>x86_64</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>14.04.16.04</td>
<td>amd64</td>
</tr>
<tr>
<td>Windows (Consumer OS)</td>
<td>7, 8.1, 10</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Windows Server (Server OS)</td>
<td>2008</td>
<td>x86, x64</td>
</tr>
<tr>
<td></td>
<td>2008r, 22012, 2012r2, 2012r2 core</td>
<td>x64</td>
</tr>
<tr>
<td></td>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>macOS</td>
<td>10.10, 10.11, 10.12, 10.13</td>
<td></td>
</tr>
</tbody>
</table>

**Install PE client tools on a managed workstation**

To use the client tools on a system other than the master, 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 master 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.

**Related information**

Create classification node groups on page 367
Create classification node groups to assign classification data to nodes.

**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 master 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 master.
- **4433** — The Puppet access client uses this port to communicate with the RBAC service running on the master.
- **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 master, 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 master.
5. Download the `pe-client-tools package` for the platform appropriate to your workstation.
6. Unpack the tarball and navigate to the `packages/<PLATFORM>` directory.
7. Use your workstation's package management tools to install the `pe-client-tools`. For example, on RHEL platforms: `rpm -Uvh pe-client-tools-<VERSION-and-PLATFORM>.rpm`

**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 master, 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 master.
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 master, 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 master.
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 master.

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**
  - *

- **Windows** — `%ProgramData%\puppetlabs\client-tools`
• User
  • *nix — ~/.puppetlabs/client-tools/
  • Windows — %USERPROFILE%\puppetlabs\client-tools

On managed client nodes where the operating system and architecture match the master, 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>• Deploying applications with Puppet Application Orchestration: workflow on page 453</td>
</tr>
<tr>
<td></td>
<td>• Running jobs with Puppet orchestrator on page 473</td>
</tr>
<tr>
<td></td>
<td>• Running Puppet on demand from the CLI on page 493</td>
</tr>
<tr>
<td></td>
<td>• Running tasks from the command line on page 509</td>
</tr>
<tr>
<td></td>
<td>• Review jobs from the command line on page 527</td>
</tr>
<tr>
<td>Puppet access</td>
<td>• Token-based authentication on page 289</td>
</tr>
<tr>
<td>Puppet code</td>
<td>• Triggering Code Manager on the command line on page 587</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>• PuppetDB CLI</td>
</tr>
</tbody>
</table>

**Related information**

**Orchestrator configuration files** on page 480

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 Puppet master or from a separate workstation, there are two types of configuration files: a global configuration file and a user-specified configuration file.

**Configuring puppet-access** on page 290

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 Puppet master.

**Installing and configuring puppet-code** on page 587

PE automatically installs and configures the puppet-code command on your masters 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.

**Installing external PostgreSQL**

By default, Puppet Enterprise includes its own database backend, PE-PostgreSQL, which is installed alongside PuppetDB. If the load on your PuppetDB node is larger than it can effectively scale to (greater than 20,000 nodes), you can install a standalone instance of PE-PostgreSQL.

In certain limited circumstances, you might choose to configure a PostgreSQL instance that's not managed by PE. Using unmanaged PostgreSQL increases complexity for maintenance and upgrades, so we recommend this configuration only for customers who can't use PE-PostgreSQL.
**Install standalone PE-PostgreSQL**

If the load on your PuppetDB node is larger than it can effectively scale to (greater than 20,000 nodes), you can install a standalone instance of PE-PostgreSQL.

**Before you begin**

You must have root access to the node on which you plan to install PE-PostgreSQL, as well as the ability to SSH and copy files to the node.

1. Prepare your `pe.conf` file by specifying parameters required for PostgreSQL.

   ```
   "puppet_enterprise::puppet_master_host": "<MASTER OF MASTERS HOSTNAME>"
   "console_admin_password": "<CONSOLE ADMIN PASSWORD>"
   "puppet_enterprise::database_host": "<PE-POSTGRESQL NODE HOSTNAME>"
   ```

2. Follow the instructions to Install using text mode (mono configuration) on page 174, running the installer with the `-c` flag.

   The installer fails halfway through, because it can't contact the database. Components that rely on the database also fail to start. The next installation run corrects this issue.

3. Copy the `pe.conf` file you created to the PE-PostgreSQL node and SSH into that node.

4. Run the installer with the `-c` flag, using the same `pe.conf` file.

5. When the installation process finishes on the PE-PostgreSQL node, SSH into the master of masters and complete the installation:

   ```
   puppet infrastructure configure; puppet agent -t;
   ```

   The master of masters is configured to use the standalone PE-PostgreSQL installation on the PE-PostgreSQL node.

**Install unmanaged PostgreSQL**

If you use Amazon RDS, or if your business requirements dictate that databases must be managed outside of Puppet, you can configure a PostgreSQL database that’s not managed by PE.

**Before you begin**

You must have:

- Postgresql 9.6 or later.
- The complete certificate authority certificate chain for the external party CA, in PEM format.
- The DNS-addressable name, username, and password for the external PostgreSQL database.

**Important:** Using unmanaged PostgreSQL increases complexity for maintenance and upgrades, so we recommend it only for customers who can’t use PE-PostgreSQL.

**Create the unmanaged PostgreSQL instance**

Create the unmanaged PostgreSQL instance, and, if you haven't already, retrieve the necessary certificate chain and credentials from your database administrator.

For example, for RDS, the root certificate is available [here](#).

**Create PE databases on the unmanaged PostgreSQL instance**

1. Log in to the unmanaged PostgreSQL instance with the client of your choice.
2. Create databases for the orchestrator, RBAC, activity service, and the node classifier.

```sql
CREATE USER "pe-puppetdb" PASSWORD '<PASSWORD>'; GRANT "pe-puppetdb" TO <ADMIN USER>
CREATE DATABASE "pe-puppetdb" OWNER "pe-puppetdb" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0;

CREATE USER "pe-orchestrator" PASSWORD '<PASSWORD>'; GRANT "pe-orchestrator" TO <ADMIN USER>
CREATE DATABASE "pe-orchestrator" OWNER "pe-orchestrator" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0;

CREATE USER "pe-activity" PASSWORD '<PASSWORD>'; GRANT "pe-activity" TO <ADMIN USER>
CREATE DATABASE "pe-activity" OWNER "pe-activity" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0;

CREATE USER "pe-classifier" PASSWORD '<PASSWORD>'; GRANT "pe-classifier" TO <ADMIN USER>
CREATE DATABASE "pe-classifier" OWNER "pe-classifier" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0;

CREATE USER "pe-rbac" PASSWORD '<PASSWORD>'; GRANT "pe-rbac" TO <ADMIN USER>
CREATE DATABASE "pe-rbac" OWNER "pe-rbac" ENCODING 'utf8' LC_CTYPE 'en_US.utf8' LC_COLLATE 'en_US.utf8' template template0;

\c "pe-rbac"
CREATE EXTENSION citext;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION plpgsql;
CREATE EXTENSION pgcrypto;

\c "pe-orchestrator"
CREATE EXTENSION citext;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION plpgsql;

\c "pe-puppetdb"
CREATE EXTENSION citext;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION plpgsql;
CREATE EXTENSION pgcrypto;

\c "pe-classifier"
CREATE EXTENSION citext;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION plpgsql;

\c "pe-activity"
CREATE EXTENSION citext;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION plpgsql;
```

Next, install PE. If you use the web-based installer, select to **Use an existing PostgreSQL instance** and specify details about your PostgreSQL configuration.
Establish SSL between PE and the unmanaged PostgreSQL instance

Before you begin

Install PE. If you use the web-based installer, select to Use an existing PostgreSQL instance and specify details about your PostgreSQL configuration.

1. Log in to the master (monolithic installation) or into the master, console, and PuppetDB nodes (split installation), and stop the agent service:

   /opt/puppetlabs/puppet/bin/puppet resource service puppet ensure=stopped

2. On the master (monolithic installation), or on the console and PuppetDB nodes (split installation), create a location to store the CA cert from the external PostgreSQL instance, for example /etc/puppetlabs/puppet/ssl/.

3. Transfer the CA cert from the unmanaged PostgreSQL instance to the directories that you created.

4. Specify ownership and permissions for the certificate and directories.

   chown -R pe-puppet:pe-puppet <PATH TO DIRECTORY>
   chmod 664 <PATH TO DIRECTORY>/<CERT NAME>

5. Modify your pe.conf file to specify database properties.

   "puppet_enterprise::profile::console::database_properties": "?
   ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=<PATH TO EXTERNAL POSTGRESQL CA CERT>"
   "puppet_enterprise::profile::puppetdb::database_properties": "?
   ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=<PATH TO EXTERNAL POSTGRESQL CA CERT>"
   "puppet_enterprise::profile::orchestrator::database_properties": "?
   ssl=true&sslfactory=org.postgresql.ssl.jdbc4.LibPQFactory&sslmode=verify-full&sslrootcert=<PATH TO EXTERNAL POSTGRESQL CA CERT>"
   "puppet_enterprise::database_ssl": true
   "puppet_enterprise::database_cert_auth": false

6. On the master (monolithic installation) or on the console node (split installation), run Puppet.

7. On the master (monolithic installation) or on the console and PuppetDB nodes (split installation), start the agent service:
   puppet resource service puppet ensure=running

External PostgreSQL options for web-based installation

During a web-based installation, if you select to Use an existing PostgreSQL instance, you must specify these details about your PostgreSQL configuration.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL server DNS name</td>
<td></td>
</tr>
<tr>
<td>Port number used by the PostgreSQL server</td>
<td>5432</td>
</tr>
<tr>
<td>PuppetDB database name</td>
<td>pe-puppetdb</td>
</tr>
<tr>
<td>PuppetDB database user</td>
<td>pe-puppetdb</td>
</tr>
<tr>
<td>PuppetDB database password</td>
<td></td>
</tr>
<tr>
<td>RBAC database name</td>
<td>pe-rbac</td>
</tr>
<tr>
<td>RBAC database user</td>
<td>pe-rbac</td>
</tr>
<tr>
<td>RBAC database password</td>
<td></td>
</tr>
<tr>
<td>Node classifier database name</td>
<td>pe-classifier</td>
</tr>
</tbody>
</table>
Uninstalling

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

Uninstall component nodes

The `puppet-enterprise-uninstaller` script is installed on the master, and in a split install, on the PuppetDB and console nodes. In order to uninstall, you must run the uninstaller on each component 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.

1. From the component 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 master, and you plan to reinstall on a component node at a later date, remove the agent certificate for that component from the master. On the master: `puppet cert clean <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**

Remove nodes on page 363

To completely remove a node from PE, you must purge the node and revoke its certificate so that it doesn’t continue to check in.

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: `run /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 master: `puppet cert clean <AGENT CERT NAME>`.

© 2020 Puppet, Inc.
**Uninstall Windows agents**
To uninstall from a Windows node, use the Windows Add or Remove Programs interface, or uninstall from the command line.

Uninstalling 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 master: puppet cert clean <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:

   ```
   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 master: puppet cert clean <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, run:

```
$ sudo ./puppet-enterprise-uninstaller -d -p
```
Upgrading

To upgrade your Puppet Enterprise deployment, you must upgrade both the infrastructure components and agents.

- **Upgrading Puppet Enterprise** on page 223
  Upgrade your PE installation as new versions become available.
- **Upgrading agents** on page 230
  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>2018.1.z</td>
<td>You're up to date!</td>
<td></td>
</tr>
<tr>
<td>2017.3.z</td>
<td>2018.1</td>
<td></td>
</tr>
<tr>
<td>2017.2.z</td>
<td>2018.1</td>
<td></td>
</tr>
<tr>
<td>2017.1.z</td>
<td>2018.1</td>
<td></td>
</tr>
<tr>
<td>2016.5.z</td>
<td>2018.1</td>
<td></td>
</tr>
<tr>
<td>2016.4.10 or later</td>
<td>2018.1</td>
<td></td>
</tr>
<tr>
<td>2016.4.9 or earlier</td>
<td>latest 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>2016.2.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016.1.z</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>latest 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>

#### Upgrade cautions

These are the major updates to recent PE versions that you should be aware of when upgrading.

**Important:** Always back up your installation before performing any upgrade.

**PostgreSQL upgrade in PE 2017.3**

PE 2017.3 and later uses PostgreSQL 9.6.

- Before upgrading, make sure you have at least 1.1 times the space used by the existing `/opt/puppetlabs/server/data/postgresql` directory. Plan for a downtime window of a couple of hours if you have a large database, and don't worry if your upgrade process seems to hang while upgrading the database—it's not hung.
• After upgrading, verify that your installation is working as expected (log into the console and check for historical reports and custom classification groups), then free disk space by cleaning up these PostgreSQL 9.4 directories:
  • /opt/puppetlabs/server/data/postgresql/9.4
  • /opt/puppetlabs/server/data/postgresql/activity/PG_9.4*
  • /opt/puppetlabs/server/data/postgresql/classifier/PG_9.4*
  • /opt/puppetlabs/server/data/postgresql/orchestrator/PG_9.4*
  • /opt/puppetlabs/server/data/postgresql/puppetdb/PG_9.4*
  • /opt/puppetlabs/server/data/postgresql/rbac/PG_9.4*

  **Important:** Don't remove any directories that start with PG_9.6.

• If you use external PostgreSQL, you must take extra steps to upgrade. For details, see Upgrading PostgreSQL on page 229.

• If you're upgrading with high availability enabled, upgrade and then forget the existing replica, and provision and enable a new replica. For details, see Upgrade with high availability enabled on page 226.

**JRuby 9k upgrade in PE 2018.1**

To support Ruby 2.3, PE 2018.1 and later changes the default setting for JRuby 9k to enabled
(puppet_enterprise::master::puppetserver::jruby_9k_enabled: true). This default differs from open source Puppet and from previous versions of PE.

• After upgrading, update any server-side installed gems or custom extensions to be compatible with Ruby 2.3 and JRuby 9k. For example, if you're using the autosign gem workflow, upgrade the gem to 0.1.3 and make sure you're not using yardoc 0.8.x. See SERVER-2161 for details.

• For information on enabling or disabling autosigning, see Autosigning certificate requests.

• If you notice issues with JRuby in PE, file a ticket rather than changing the default parameter to avoid issues when this setting is eventually deprecated.

**MCollective deprecation in PE 2018.1**

PE 2018.1 and later no longer installs MCollective by default. If you have an existing PE installation that relies on MCollective and you upgrade by installing the new version and then moving agents over, you must enable MCollective when installing 2018.1, prior to migrating agents.

**Related information**

  **Backing up and restoring Puppet Enterprise** on page 753
  Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

  **Setting up MCollective** on page 209
  Because MCollective does not install with PE you must enable it.

**Test modules before upgrade**

To ensure that your modules will 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 a monolithic installation**

To upgrade a monolithic installation, run the text-based PE installer on your master or master of masters, and then upgrade any additional components.

**Before you begin**

Back up your installation.

If you encounter errors during upgrade, you can fix them and run the installer again.

1. **Download** the tarball appropriate to your operating system and architecture.
2. **Unpack** the installation tarball:

   ```
   tar -xf <tarball>
   ```

   You need about 1 GB of space to untar the installer.

3. From the installer directory, run the installer:

   ```
   sudo ./puppet-enterprise-installer
   ```

   **Note:** 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`.

4. If you have compile masters, upgrade them.

   SSH into each compile master and run:

   ```
   /opt/puppetlabs/puppet/bin/curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<PUPPET MASTER FQDN>:8140/packages/currentupgrade.bash
   ```

   ```
   sudo bash
   ```

© 2020 Puppet, Inc.
5. If you have hubs and spokes, upgrade them.
   SSH into each hub and spoke and run:

   ```bash
   /opt/puppetlabs/puppet/bin/curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<PUPPET MASTER FQDN>:8140/packages/current/upgrade.bash | sudo bash
   ```

6. Upgrade these additional infrastructure components.
   - Agents
   - client tools — Install the appropriate version of client tools that matches the version you upgraded to.

Related information

Backing up and restoring Puppet Enterprise on page 753
Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

Upgrade with high availability enabled

If you have high availability enabled, your upgrade path differs depending on what version you're upgrading from. Regardless of upgrade method, the replica is temporarily unavailable to serve as backup during this process, so you should time your upgrade to minimize risk.

Before you begin
Back up your installation.

Related information

Backing up and restoring Puppet Enterprise on page 753
Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

Upgrade from pre-2017.3 with high availability enabled

Upgrading from PE versions earlier than 2017.3 requires forgetting and re-creating your replica in order to update to the latest PostgreSQL version.

1. Upgrade PE.
2. Forget your existing replica.
3. Provision a new replica.
4. Enable the new replica.

Related information

Forget a replica on page 273
Forgetting a replica cleans up classification and database state, preventing degraded performance over time.

Provision a replica on page 269
Provisioning a replica duplicates specific components and services from the master to the replica.

Enable a replica on page 270
Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.

Upgrade from 2017.3 or later with high availability enabled

Upgrading from 2017.3 or later requires upgrading and reinitializing your replica.

1. Upgrade PE.
2. On your replica, run the upgrade script:
   ```
curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<MASTER_HOST>:8140/packages/current/upgrade.bash | bash
   ```
3. Verify that master and replica services are operational:
   ```
   /opt/puppetlabs/bin/puppet-infra status
   ```
4. If your replica reports errors, reinitialize the replica:
   ```
   /opt/puppetlabs/bin/puppet-infra reinitialize replica -y
   ```

Upgrade a split or large environment installation

To upgrade a split or large environment installation, run the text-based installer on each infrastructure node in your environment, and then upgrade any additional components.

**Before you begin**

Back up your installation.

**Note:** You must upgrade the components in the order specified.

If you encounter errors during upgrade, you can fix them and run the installer again.

**Related information**

[Backing up and restoring Puppet Enterprise](#) on page 753

Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

**Upgrade the master**

Upgrading the master is the first step in upgrading a split or large environment installation.

1. **Download** the tarball appropriate to your operating system and architecture.
2. Unpack the installation tarball: `tar -xf <tarball>`
   
   You need about 1 GB of space to untar the installer.
3. Stop the agent service: `sudo puppet resource service puppet ensure=stopped`
4. From the installer directory, run the installer: `sudo ./puppet-enterprise-installer`
   
   **Note:** 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. When upgrade completes, transfer the installer and the `pe.conf` file located at `/etc/puppetlabs/enterprise/conf.d/` to the next server that you're upgrading a component on.

**Upgrade PuppetDB**

In a split installation, after you upgrade the master, you're ready to upgrade PuppetDB.

1. Unpack the installation tarball: `tar -xf <tarball>`
   
   You need about 1 GB of space to untar the installer.
2. Stop the agent service: `sudo puppet resource service puppet ensure=stopped`
3. From the installer directory, run the installer: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`
4. When upgrade completes, transfer the installer and the `pe.conf` file located at `/etc/puppetlabs/enterprise/conf.d/` to the next server that you're upgrading a component on.
**Upgrade the console**
In a split installation, after you upgrade the master and PuppetDB, you're ready to upgrade the console.

1. Unpack the installation tarball:
   ```bash
tar -xf <tarball>
```
   You need about 1 GB of space to untar the installer.
2. Stop the agent service:
   ```bash
   sudo puppet resource service puppet ensure=stopped
   ```
3. From the installer directory, run the installer:
   ```bash
   sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>
   ```

**Run Puppet on infrastructure nodes**
To complete a split upgrade, run Puppet on all infrastructure nodes in the order that they were upgraded.

1. Run Puppet on the master node.
2. Run Puppet on the PuppetDB node.
3. Run Puppet on the console node.

**Upgrade remaining infrastructure nodes**
After the main components of your infrastructure are upgraded, you must upgrade any additional infrastructure nodes, such as compile masters, hubs, and spokes.

1. If you have compile masters, upgrade them.
   
   SSH into each compile master and run:
   ```bash
   /opt/puppetlabs/puppet/bin/curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<PUPPET MASTER FQDN>:8140/packages/current/upgrade.bash | sudo bash
   ```
2. If you have hubs and spokes, upgrade them.
   
   SSH into each hub and spoke and run:
   ```bash
   /opt/puppetlabs/puppet/bin/curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<PUPPET MASTER FQDN>:8140/packages/current/upgrade.bash | sudo bash
   ```
3. Upgrade these additional infrastructure components.
   - Agents
   - client tools — Install the appropriate version of client tools that matches the version you upgraded to.

**Migrate from a split to a monolithic installation**
Split installations, where the master, console, and PuppetDB are installed on separate nodes, are deprecated. Migrate from an existing split installation to a monolithic installation—with or without compilers—and a standalone PE-PostgreSQL node.

**Before you begin**
You must be running a version of PE on all infrastructure nodes that includes the `puppet infrastructure run` command. To verify that this command is available on your systems, run `puppet infrastructure run --help`.

The `puppet infrastructure run` command leverages built-in plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your master 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 master. For more information, see [OpenSSH configuration options](#).

© 2020 Puppet, Inc.
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.

**Important:** The migration command used in this task automates a number of manual steps, including editing pe.conf, unpinning and uninstalling packages from affected infrastructure nodes, and running Puppet multiple times. Treat this process as you would any major migration by thoroughly testing it in an environment that's as similar to your production environment as possible.

1. On your master, verify that pe.conf contains correct information for console_host, puppetdb_host, and database_host.
   The migration command uses this information to correctly migrate these nodes.
   **Note:** If your split PE installation includes multiple standalone PuppetDB nodes, the migration command will fail with an error.
2. Make sure that the master can connect via SSH to your console node, PuppetDB node, and (if present) standalone PE-PostgreSQL node.
3. On your master logged in as root, run puppet infrastructure run migrate_split_to_mono
   You can specify this optional parameter:
   • tmpdir — Path to a directory to use for uploading and executing temporary files.

After completion, your master is running the console and PuppetDB services and you can retire or repurpose the old console node. If you did not start with a standalone PE-PostgreSQL node, your old PuppetDB node now functions in that capacity. If you did start with a standalone PE-PostgreSQL node, it continues to function in that capacity and you can retire or repurpose the old PuppetDB node.

**Related information**

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 Puppet master.

**Upgrading PostgreSQL**

If you use the default PE-PostgreSQL database installed alongside PuppetDB, you don't have to take special steps to upgrade Postgresql. However, if you have a standalone PE-PostgreSQL instance, or if you use a PostgreSQL instance not managed by PE, you must take extra steps to upgrade PostgreSQL.

You must upgrade a standalone PE-PostgreSQL instance each time you upgrade PE. To upgrade a standalone PE-PostgreSQL instance, simply run the installer on the PE-PostgreSQL node first, then proceed with upgrading the rest of your infrastructure.

You must upgrade a PostgreSQL instance not managed by PE only when there's an upgrade to PostgreSQL in PE, which occurred most recently in 2017.3. To upgrade an unmanaged PostgreSQL instance, use one of these methods:

• Back up databases, wipe your old PostgreSQL installation, install the latest version of PostgreSQL, and restore the databases.
• Back up databases, set up a new node with the latest version of PostgreSQL, restore databases to the new node, and reconfigure PE to point to the new database_host.
• Run pg_upgrade to get from the older PostgreSQL version to the latest version.

**Note:** If you're upgrading a split installation of a PE version earlier than 2016.4.3 with an external PostgreSQL instance, you must upgrade with the --force flag, for example:

```
/opt/puppetlabs/puppet/bin/puppet infrastructure configure --detailed-exitcodes --modulepath=/opt/puppetlabs/server/data/enterprise/modules --no-noop --upgrade-from=<PREVIOUS PE VERSION> --force
```

Upgrading this configuration without the force flag causes the upgrade to hang while searching for the external database.
Text mode installer options

When you run the installer in text mode, you can use the `-c` option to specify the full path to an existing `pe.conf` file. You can pair these additional options with the `-c` option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D</code></td>
<td>Display debugging information</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Run in quiet mode. The installation process isn’t displayed. If errors occur during the installation, the command quits with an error message.</td>
</tr>
<tr>
<td><code>-y</code></td>
<td>Run automatically using the <code>pe.conf</code> file at <code>/etc/puppetlabs/enterprise/conf.d/</code>. If the file is not present or is invalid, installation or upgrade fails.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Display verbose debugging information.</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Display help information.</td>
</tr>
<tr>
<td><code>force</code></td>
<td>For upgrades only, bypass PostgreSQL migration validation. This option must appear last, after the end-of-options signifier (<code>--</code>), for example <code>sudo ./puppet-enterprise-installer -c pe.conf -- --force</code></td>
</tr>
</tbody>
</table>

Checking for updates

To see the version of PE you're currently using, run `puppet --version` on the command line. Check the PE download site to find information about the latest maintenance release.

**Note:** By default, the master checks for updates whenever the `pe-puppetserver` service restarts. As part of the check, it passes some basic, anonymous information to Puppet servers. You can optionally disable update checking.

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, first upgrade your master and verify that the master 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 master but use an older agent version that is still compatible with the new master, 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 master, 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 master upgrade.

Related information

Upgrading Puppet Enterprise on page 223
Upgrade your PE installation as new versions become available.

**Upgrade *nix or Windows agents using the puppet_agent module**

The puppetlabs-puppet_agent module, available from the Forge, enables you to upgrade multiple agents at once. The module handles all the latest version to version upgrades.

When upgrading agents, first test the upgrade on a subset of agents, and after you verify the upgrade, upgrade remaining agents.

1. On your master, download and install the puppetlabs-puppet_agent module:
   ```bash
   puppet module install puppetlabs-puppet_agent
   ```
2. Configure the master to download the agent version you want to upgrade.
   a) In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) On the **Configuration** 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 master.
   c) Click **Add class** and commit changes.
   d) On your master, run to configure the newly assigned class:
   ```bash
   puppet agent -t
   ```
   The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.
3. Click **Classification**, 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`
6. Still in the agent upgrade group, click the **Configuration** tab, and in the **Add new class** field, 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://<MASTER_HOSTNAME>:8140/<Prefix>`.

© 2020 Puppet, Inc.
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 <code>puppet-agent</code> package version to install, for example 5.3.3.</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to <code>auto</code> to install the same agent version that is installed on your master.</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 a *nix or Windows agent 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 master.

**Note:** If you encounter SSL errors during the upgrade process, ensure your agent’s OpenSSL is up to date and matches the master’s version. You can check the master’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 an upgrade an individual *nix agent using a script.

1. Configure the master to download the agent version you want to upgrade.
   a) In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.
   b) On the **Configuration** 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 master.
   c) Click **Add class** and commit changes.
   d) On your master, run 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

     ```bash
     /opt/puppetlabs/puppet/bin/curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash
     ```
   - , , and

     ```bash
     curl --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem https://<MASTER HOSTNAME>:8140/packages/current/install.bash | sudo bash
     ```

   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 `<MASTER_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the master. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. Stop the Puppet service, the PXP agent service, and the MCollective service.
2. On the Windows agent, run the install script:

   ```bash
   
   3. Verify that Puppet runs are complete.
4. Restart the Puppet service, the PXP agent service, and the MCollective service.

Upgrading the agent independent of PE
You can optionally upgrade the agent to a newer version than the one packaged with your current PE installation.

For details about Puppet agents versions that are tested and supported for PE, see the PE component version table.

The agent version is specified on a platform-by-platform basis in the PE Master node group, in any `pe_repo::platform` class, using the `agent_version` parameter.

When you install new nodes or upgrade existing nodes, the agent install script installs the version of the agent specified for its platform class. If a version isn’t specified for the node’s platform, the script installs the default version packaged with your current version of PE.

Note: To install nodes without internet access, download the agent tarball for the version you want to install, as specified using the `agent_version` parameter.

The platform in use on your master requires special consideration. The agent version used on your master must match the agent version used on other infrastructure nodes, including compilers, replicas, and MCollective hubs and spokes, otherwise your master won’t compile catalogs for these nodes.

To keep infrastructure nodes synced to the same agent version, if you specify a newer `agent_version` for your master platform, you must either:

• (Recommended) Upgrade the agent on your master—and any existing infrastructure nodes—to the newer agent version. You can upgrade these nodes by running the agent install script.

• Manually install the older agent version used on your master on any new infrastructure nodes you provision. You can’t install these nodes using the agent install script, because the script uses the agent version specified for the platform class, instead of the master’s current agent version. Manual installation requires configuring `puppet.conf`, DNS alt names, CSR attributes, and other relevant settings.

Related information
Component versions in recent PE releases on page 22
This is a historical overview of which components are in Puppet Enterprise (PE) versions, dating back to the previous long term supported (LTS) release.

**Upgrade 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 upgrade using a script.

**Before you begin**

Download the appropriate agent tarball from an internet-connected system.

1. On your master, 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. Run Puppet: `puppet agent -t`
3. Follow the steps to Upgrade a *nix or Windows agent using a script on page 232.

**Configuring Puppet Enterprise**

**Important:** 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.
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 own certificate to the whitelist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

**Methods for configuring Puppet Enterprise** on page 235

There are three main methods for configuring PE: using the console, adding a key to Hiera, or editing pe.conf. For the most part, you can choose to use whatever method you want, but sometimes one might be better than another depending on the situation. In general, try to stay as consistent as possible.

**Important:** When you enable high availability, you must use or pe.conf only — not the console — to specify configuration parameters. Using pe.conf or ensures that configuration is applied to both your master and replica.

**Related information**

- Configuring and tuning Puppet Server on page 237
After you've installed Puppet Enterprise, optimize it for your environment by configuring and tuning Puppet Server settings as needed.

**Configuring and tuning PuppetDB** on page 245
After you've installed Puppet Enterprise, optimize it for your environment by configuring and tuning PuppetDB configuration as needed.

**Configuring and tuning the console** on page 242
After installing Puppet Enterprise, you can change product settings to customize the console's behavior, adjust to your team's needs, and improve performance.

**Configuring and tuning orchestration** on page 247
After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.

**Configuring Java arguments for Puppet Enterprise** on page 251
You might need to increase the Java Virtual Machine (JVM) memory allocated to Java services or ActiveMQ to improve performance in your Puppet Enterprise (PE) deployment.

### Configure settings using the console

Using the console to change parameters is the most user-friendly way of configuring Puppet Enterprise (PE) because you can use a graphical interface in order to make changes.

Changes in the console will 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.

To configure settings in the console, do the following:

1. In the console, click **Classification**, and select the node group that contains the class you want to work with.
2. On the **Configuration** tab, find the class you want to work with, select the **Parameter name** from the list and edit its value.
   
   If you wanted to change the number used to identify the port your console is on from the default 443 to 500, change the parameter value in the following:

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>puppet_enterprise::profile</code></td>
<td><code>console_ssl_listen_port</code></td>
<td>[500]</td>
</tr>
</tbody>
</table>

3. Click **Add parameter** and commit changes.
4. On the nodes hosting the master and console, run Puppet.

### Related information

**Preconfigured node groups** on page 375
Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

### 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.

#### Before you begin

For more information on how to use Hiera, see the [Hiera docs](https://hiera.readthedocs.io/en/latest/).

Changes in Hiera will override `pe.conf`, but not 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, do the following:

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.
   If you wanted 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. Compile and view your overrides.

Related information
Preconfigured node groups on page 375
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.

Hiera data and the console always override pe.conf. Configure settings using pe.conf when you want to:
• Quickly access settings during installation.
• Configure for high availability.

Keep in mind, you might notice a nodes/ directory with files named after nodes in your PE infrastructure. These files are created automatically for you to keep your PE configuration data in sync with your user data.

1. Open your pe.conf file on your master.

   /etc/puppetlabs/enterprise/conf.d/pe.conf

2. Add the parameter and new value you want to set.
   If you wanted 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
   If PE services are stopped, run puppet infrastructure configure.

Configuring and tuning Puppet Server
After you’ve installed Puppet Enterprise, optimize it for your environment by configuring and tuning Puppet Server settings as needed.

See the configuration methods docs for information on how to configure settings.

Related information
Tuning monolithic installations on page 254
Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

**Configure settings with Hiera** on page 236

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.

**Increase the Java heap size for PE Java services** on page 251

The Java heap size is the Java Virtual Machine (JVM) memory allocated to Java services in Puppet Enterprise (PE). You can use any configuration method you choose. We will use the console to change the Java heap size for console services, Puppet Server, orchestration services, or PuppetDB in the examples below.

**Configure ulimit for PE services** on page 253

The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults will not be high enough for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

### Tune the maximum number of JRuby instances

The `jruby_max_active_instances` setting controls the maximum number of JRuby instances to allow on the Puppet Server.

The default used in PE is the number of CPUs - 1, expressed as `$::processorcount - 1`. One instance is the minimum value and four instances is the maximum value. Four JRuby instances work for most environments.

Increasing the JRuby instances increases the amount of RAM used by `pe-puppetserver`. When you increase JRuby instances, increase the heap size. We conservatively estimate that a JRuby process uses 512MB of RAM.

1. To change the number of instances using Hiera, add the following to your default `.yaml` file and set the desired number of instances:

   ```yaml
   puppet Enterprise::master::puppetserver::jruby_max_active_instances: <number of instances>
   ```

2. Compile and view overrides.

### 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 here: `$puppetserver_jruby_puppet_ruby_load_path = ['/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib']`.

1. To change the path to a different array in `pe.conf`, add the following to your `pe.conf` file on your master and set your new load path parameter:

   ```yaml
   puppet Enterprise::master::puppetserver::<puppetserver_jruby_puppet_ruby_load_path>
   ```

2. Run `puppet agent -t`.

Note that if you change the `libdir` you must also change the `vardir`.

### Tune the maximum requests per JRuby instance

The `max_requests_per_instance` setting determines the maximum number of requests per instance of a JRuby interpreter before it is killed.

The appropriate value for this parameter depends on how busy your servers are and how much you are affected by a memory leak. By default, `max_requests_per_instance` is set to 100,000 in PE.

When a JRuby interpreter is killed, all of its memory is reclaimed and it is replaced in the pool with a new interpreter. This prevents any one interpreter from consuming too much RAM, mitigating Puppet code memory leak issues and keeping Puppet Server up.
Starting a new interpreter has a performance cost, so set the 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.

1. To increase `max_requests_per_instance` using Hiera, add the following code to your default `.yaml` and set the desired number of requests:

   ```yaml
   puppet_enterprise::master::puppetserver::jruby_max_requests_per_instance: <number of requests>
   ```

2. Compile and view overrides.

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.

1. To enable or disable cached data using Hiera, add the following to your default `.yaml` file and set the parameter to `true` or `false`.

   ```yaml
   puppet_enterprise::master::puppetserver::jruby_environment_class_cache_enabled: <true OR false>
   ```

2. Compile and view overrides.

Changing the `environment_timeout` setting

The `environment_timeout` setting controls how long the master 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 master 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`.

Change the `environment_timeout` setting using `pe.conf`, add the following to your `pe.conf` file on your master, then run `puppet agent -t`:

```bash
puppet_enterprise::master:: <environment_timeout>
```

For more information, see [Environments limitations](https://puppet.com/docs/puppet-enterprise/pe-manual/environments-limitations).

Add certificates to the puppet-admin certificate whitelist

Change the puppet-admin certificate whitelist as needed.

1. To modify whitelist certificates using `pe.conf`, insert the following in your `pe.conf` file on your master and add certificates.

   ```bash
   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, Inc. 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 **Classification**, and select the node group that contains the class you want to work with.
2. On the **Configuration** tab, find the `puppet_enterprise::profile` class and find the `check_for_updates` parameter from the list and change its value to `false`.

```
puppet_enterprise::profile::master::check_for_updates: false
```

3. Click **Add parameter** and commit changes.
4. On the nodes hosting the master 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 and external Certificate Authority 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](#) and the Related topics below.

**Related information**

- [Viewing and managing Puppet Server metrics](#) on page 340
Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

pe-puppet-server.conf settings

This file contains Puppet Server settings specific to Puppet Enterprise, with all settings wrapped in a jruby-puppet section.

gen-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

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-queued-requests
Optional. Sets the maximum number of requests that may be queued waiting to borrow a from the pool. Once this limit is exceeded, a 503 Service Unavailable response is returned for all new requests until the queue drops below the limit.

If max-retry-delay is set to a positive value, then the 503 response includes a Retry-After header indicating a random sleep time after which the client may retry the request.

Note: Don't use this solution if your managed infrastructure includes a significant number of agents older than Puppet 5.3. Older agents treat a 503 response as a failure, which ends their runs, causing groups of older agents to schedule their next runs at the same time, creating a thundering herd problem.
Default: 0

max-retry-delay
Optional. Sets the upper limit in seconds for the random sleep set as a Retry-After header on 503 responses returned when max-queued-requests is enabled.
Default: 1800

jruby_max_active_instances
Controls the maximum number of JRuby instances to allow on the Puppet Server.
Default: 4

max_requests_per_instance
Sets the maximum number of requests per instance of a JRuby interpretor before it is killed.
Default: 100000

ruby-load-path
Sets the Puppet configuration directory's path. The agent's libdir value is added by default.
Default: '/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib'
Configuring and tuning the console

After installing Puppet Enterprise, you can change product settings to customize the console's behavior, adjust to your team's needs, and improve performance.

**Related information**
- Disable update checking on page 240
  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, Inc. to get update information. You can optionally turn this off.
- Configuring Java arguments for Puppet Enterprise on page 251
  You might need to increase the Java Virtual Machine (JVM) memory allocated to Java services or ActiveMQ to improve performance in your Puppet Enterprise (PE) deployment.
- Configure ulimit for PE services on page 253
  The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults will not be high enough for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

**Configure the PE console and console-services**

Configure the behavior of the console and console-services, as needed.

**Note:** You cannot use the console to configure non-profile classes, such as puppet_enterprise::api_port and puppet_enterprise::console_services::no_longer_reporting_cutoff.

**Related information**
- Running Puppet on nodes on page 364
  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.

**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 master.
  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 may 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 may be valid. May be set to "0" to not expire before the maximum token lifetime.
  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::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
  Integer representing the number of days to wait before pruning the size of the classifier database. If you set the value to "0", the node classifier service is never pruned.

puppet_enterprise::profile::console::classifier_node_check_in_storage
  "true" to store an explanation of how nodes match each group they're classified into, or "false".
  Default: false

puppet_enterprise::profile::console::display_local_time
  "true" to display timestamps in local time, with hover text showing UTC time, or "false" to show timestamps in UTC time.
  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.

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.

  Tip: Some of these settings are in a profile class and can be edited in the console.

Customize the HTTPS redirect target URL
  By default, the redirect target URL is the same as the FQDN of your master, but you can customize this redirect URL.

  1. To change the target URL with the console, click Classification, and select the node group that contains the class you want to work with.
2. On the **Configuration** tab, find the `puppet_enterprise::profile::console::proxy::http_redirect` class, find the `server_name` parameter from the list and change its value to the desired server.

   `puppet_enterprise::profile::console::proxy::http_redirect::server_name: example-URL.corp.net`

3. Click **Add parameter** and commit changes.

4. On the nodes hosting the master 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. Edit your Hiera data file to disable HTTP redirect.

   `puppet_enterprise::profile::console::proxy::http_redirect::enable_http_redirect: false`

2. Run Puppet on your master.

**Tuning the PostgreSQL buffer pool size**
If you are experiencing performance issues or instability with the console, adjust the buffer memory settings for PostgreSQL.

The most important PostgreSQL memory settings for PE are `shared_buffers` and `work_mem`.

1. In the console, click **Classification**, and in the PE Infrastructure group, select the PE Database group.

2. On the **Configuration** tab, specify parameters as needed and commit changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shared_buffers</code></td>
<td>Set at about 25 percent of your hardware's RAM.</td>
</tr>
<tr>
<td><code>work_mem</code></td>
<td>In large or complex deployments, increase the value from the default 1MB.</td>
</tr>
</tbody>
</table>

3. Restart the PostgreSQL server: `sudo /etc/init.d/pe-postgresql restart`

**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.

On your master, edit `/etc/puppetlabs/puppet/hiera.yaml` to add:

```
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, you might need to add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

If your environment is configured for high availability, you must also update `hiera.yaml` on your replica.
Configuring and tuning PuppetDB

After you've installed Puppet Enterprise, optimize it for your environment by configuring and tuning PuppetDB configuration as needed.

Additional 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.

Related information

- Tuning monolithic installations on page 254
  Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

- Configure settings with Hiera on page 236
  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.

- Configure ulimit for PE services on page 253
  The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults will not be high enough for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

- Increase the Java heap size for PE Java services on page 251
  The Java heap size is the Java Virtual Machine (JVM) memory allocated to Java services in Puppet Enterprise (PE). You can use any configuration method you choose. We will use the console to change the Java heap size for console services, Puppet Server, orchestration services, or PuppetDB in the examples below.

Configure agent run reports in the console

By default, every time Puppet runs, the master generates agent run reports and submits them to PuppetDB. These agent run reports can be enabled or disabled in the console.

1. Click Classification, and in the PE Infrastructure group, select the PE Master group.
2. On the Configuration tab, locate or add the puppet_enterprise::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.

Configure agent run reports in Hiera

By default, every time Puppet runs, the master generates agent run reports and submits them to PuppetDB. These agent run reports can be enabled or disabled in Hiera.

1. Edit your Hiera default .yaml file.
2. Set the report_processor_ensure setting to present (enabled) or absent (disabled), as shown in the following code:

   puppet_enterprise::profile::master::puppetdb::report_processor_ensure: <PRESENT or ABSENT>

Configure command processing threads

The command_processing_threads setting defines how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.

This setting defaults to half the number of cores in your system.
Set the number of command processing threads by editing your Hiera default .yaml file to add the following code:

```
puppet_enterprise::puppetdb::command_processing_threads: <NUMBER OF THREADS>
```

### Configuring broker memory

The `memory-usage` parameter sets the maximum amount of memory in megabytes available for the PuppetDB ActiveMQ broker.

Tuning this setting involves writing Puppet code. See the [PuppetDB documentation on configuring memory-usage](#) and [Playing nice with the PuppetDB module](#).

### Configure node-purge-ttl

Use this parameter to set the "time-to-live" value before PE automatically deletes nodes that have been deactivated or expired. This will also delete all facts, catalogs, and reports for the relevant nodes.

Edit the Hiera default .yaml file and set the `node_purge_ttl` setting.

For example, to specify a value of 14 days:

```
puppet_enterprise::profile::puppetdb::node_purge_ttl: '14d'
```

To set time using other units, use the following suffixes:

- **d** - days
- **h** - hours
- **m** - minutes
- **s** - seconds
- **ms** - milliseconds

### 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 `psql` 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 blacklisted facts

Use the `facts_blacklist` to exclude facts from being stored in the PuppetDB database.

Edit the Hiera default .yaml file and set the `facts_blacklist` setting.

For example, to prevent the `system_uptime` and `mountpoints` facts from being stored in PuppetDB:

```
puppet_enterprise::puppetdb::database_ini::facts_blacklist: - 'system_uptime'
```

© 2020 Puppet, Inc.
Configuring and tuning orchestration

After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.

Related information

Tuning monolithic installations on page 254
Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

Configure settings with Hiera on page 236
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.

Configure ulimit for PE services on page 253
The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults will not be high enough for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

Configure the orchestrator and pe-orchestration-services

There are several parameters you can add to configure the behavior of the orchestrator and pe-orchestration-services.

1. In the console, click Classification, and in the PE Infrastructure group, select the PE Orchestrator group.
2. On the **Configuration** tab, locate the `puppet_enterprise` class indicated and add any of the following parameters and values as needed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>puppet_enterprise::profile::agent::pxp_enabled</code></td>
<td>Disable or enable the PXP service. Set to <code>true</code> or <code>false</code>. If you disable this setting you can’t use the orchestrator or the Run Puppet button in the console. Enabled (<code>true</code>) by default.</td>
</tr>
<tr>
<td><code>puppet_enterprise::profile::orchestrator::global_concurrent_compiles</code></td>
<td>An integer that determines how many concurrent compile requests can be outstanding to the master, across all orchestrator jobs. The default value is &quot;8&quot;.</td>
</tr>
<tr>
<td><code>puppet_enterprise::profile::orchestrator::job_prune_threshold</code></td>
<td>Integer representing the days after which job reports should be removed. Defaults to &quot;30&quot; days.</td>
</tr>
<tr>
<td><code>puppet_enterprise::profile::orchestrator::pcp_timeout</code></td>
<td>An agent needs to connect to the PCP broker in order to do Puppet runs via the orchestrator. Set an integer to specify how much time should pass before the connection times out. The orchestrator defaults to &quot;30&quot; seconds. If the agent can’t connect to the broker in that time frame, the run will timeout. Disables or enable orchestration services. Set to <code>true</code> or <code>false</code>. Enabled (<code>true</code>) by default.</td>
</tr>
<tr>
<td><code>puppet_enterprise::profile::orchestrator::task_concurrency</code></td>
<td>Integer representing the number of tasks that can run at the same time.Defaults to &quot;250&quot; tasks.</td>
</tr>
<tr>
<td><code>puppet_enterprise::profile::orchestrator::use_application_services</code></td>
<td>Disable or enable application management. Set to <code>true</code> or <code>false</code>. Disabled (<code>false</code>) by default.</td>
</tr>
<tr>
<td><code>puppet_enterprise::pxp_agent::ping_interval</code></td>
<td>Controls how frequently PXP agents will ping PCP brokers. If the agents don’t receive responses, they will attempt to reconnect. Defaults to &quot;120&quot; seconds.</td>
</tr>
<tr>
<td><code>puppet_enterprise::pxp_agent::pxp_logfile</code></td>
<td>A string that represents the path to the PXP agent log file. Change as needed. By default, the log files are located at:</td>
</tr>
<tr>
<td></td>
<td>* *nix: <code>/var/log/puppetlabs/pxp-agent/pxp-agent.log</code></td>
</tr>
<tr>
<td></td>
<td>Windows: C:\Program Data\PuppetLabs\pxp-agent\var\log\pxp-agent.log</td>
</tr>
<tr>
<td><code>puppet_enterprise::pxp_agent::spool_dir_purge_ttl</code></td>
<td>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). Controls how long tasks should be cached after use. By default, unused tasks are purged after 2 weeks. You can declare time in minutes (30m), hours (2h), and days (14d).</td>
</tr>
<tr>
<td><code>puppet_enterprise::pxp_agent::task_cache_dir_purge_ttl</code></td>
<td></td>
</tr>
</tbody>
</table>

3. Click **Add Parameter** as needed, and commit changes.

4. On the node hosting the master, run Puppet.

**Related information**

[Running Puppet on nodes](#) on page 364
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.

**Configure PXP agent log file location**

Use the PXP agent log file to debug issues with the Puppet orchestrator.

By default the log files are at `/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 these locations with Hiera.

Add the following parameter to your Hiera configuration:

```
puppet_enterprise::pxp_agent::pxp_logfile: '<PATH TO LOG FILE>'
```

**Correct ARP table overflow**

In larger deployments that use MCollective or the PCP broker, you may 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, this can lead to an increase of extra network traffic (increasing CPU load on the broker and network latency) to restore them.

A typical log message will resemble the following:

```
[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_thres3=4096
net.ipv4.neigh.default.gc_thres3=4096
# Start aggressively clearing the table at this threshold
net.ipv6.neigh.default.gc_thres2=2048
net.ipv4.neigh.default.gc_thres2=2048
# Don't clear any entries until this threshold
net.ipv6.neigh.default.gc_thres1=1024
net.ipv4.neigh.default.gc_thres1=1024
```

**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 master 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 that `pe_repo` can access agent tarballs.

In `pe.conf`, Hiera, or the console, 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. For example, to specify these settings in `pe.conf`, add these lines:

```yaml
"pe_repo::http_proxy_host": "proxy.example.vlan",
"pe_repo::http_proxy_port": 8080
```

**Tip:** To test proxy connections to `pe_repo`, run:

```bash
curl -x http://proxy.example.vlan:8080 -I https://pm.puppetlabs.com
```

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, specify values for the `http_proxy_host` and `http_proxy_port` settings in `/etc/puppetlabs/puppet/puppet.conf`, for example:

```bash
http_proxy_host = proxy.example.vlan
http_proxy_port = 8080
```

For more information about HTTP proxy host options, see the Puppet configuration reference.

Setting a proxy for Code Manager traffic

Code Manager has its own set of proxy configuration options which 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:** In order to set a proxy for Code Manager connections, you must use an HTTP URL for your r10k remote and for all Puppetfile module entries.

To configure Code Manager to use a proxy for all HTTP connections, including both Git and the Forge, do one of the following:

- In the console, in the `puppet_enterprise::profile::master` class of the PE Master node group, specify a value for the `r10k_proxy` parameter.
- In Hiera, add this key:

  ```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
curl -x http://proxy.example.vlan:8080 -I https://github.com

curl -x http://proxy.example.vlan:8080 -I https://forgeapi.puppet.com
```

For detailed information about configuring proxies for Code Manager traffic, see the Code Manager documentation.

**Related information**

- Configuring proxies on page 583

© 2020 Puppet, Inc.
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 Java arguments for Puppet Enterprise

You might need to increase the Java Virtual Machine (JVM) memory allocated to Java services or ActiveMQ to improve performance in your Puppet Enterprise (PE) deployment.

**Important:** When you enable high availability, you must use `or pe.conf only — not the console — to specify configuration parameters. Using `pe.conf` or ensures that configuration is applied to both your master and replica.

**Related information**

- **Tuning monolithic installations** on page 254
  Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.
- **Configure settings with Hiera** on page 236
  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.
- **Methods for configuring Puppet Enterprise** on page 235
  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 own certificate to the whitelist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

## Increase the Java heap size for PE Java services

The Java heap size is the Java Virtual Machine (JVM) memory allocated to Java services in Puppet Enterprise (PE). You can use any configuration method you choose. We will use the console to change the Java heap size for console services, Puppet Server, orchestration services, or PuppetDB in the examples below.

**Note:** Ensure that you have sufficient free memory before increasing the memory that is used by a service. The increases shown below are only examples.

1. In the console, click **Classification**. In the **PE Infrastructure** node group, select the appropriate node group.

<table>
<thead>
<tr>
<th>Service</th>
<th>Node group</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-console-services</td>
<td>PE Console</td>
<td>puppet_enterprise::profile::console</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>PE Master</td>
<td>puppet_enterprise::profile::master</td>
</tr>
<tr>
<td>pe-orchestration-services</td>
<td>PE Orchestator</td>
<td>puppet_enterprise::profile::orchestrator</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>PE PuppetDB</td>
<td>puppet_enterprise::profile::puppetdb</td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>PE ActiveMQ Broker</td>
<td>puppet_enterprise::profile::amq::br</td>
</tr>
</tbody>
</table>

2. Click **Configuration** and scroll down to the appropriate class.
3. Click the **Parameter name** list and select `java_args`. Increase the heap size by replacing the parameter with the appropriate JSON string.

<table>
<thead>
<tr>
<th>Service</th>
<th>Default heap size</th>
<th>New heap size</th>
<th>JSON string</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-console-services</td>
<td>256 MB</td>
<td>512 MB</td>
<td><code>{&quot;Xmx&quot;: &quot;512m&quot;, &quot;Xms&quot;: &quot;512m&quot;}</code></td>
</tr>
<tr>
<td>Puppet Server</td>
<td>2 GB</td>
<td>4 GB</td>
<td><code>{&quot;Xmx&quot;: &quot;4096m&quot;, &quot;Xms&quot;: &quot;4096m&quot;}</code></td>
</tr>
<tr>
<td>orchestration-services</td>
<td>192 MB</td>
<td>1000 MB</td>
<td><code>{&quot;Xmx&quot;: &quot;1000m&quot;, &quot;Xms&quot;: &quot;1000m&quot;}</code></td>
</tr>
<tr>
<td>PuppetDB</td>
<td>256 MB</td>
<td>512 MB</td>
<td><code>{&quot;Xmx&quot;: &quot;512m&quot;, &quot;Xms&quot;: &quot;512m&quot;}</code></td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>512 MB</td>
<td>1024 MB</td>
<td><code>{&quot;Xmx&quot;: &quot;1024m&quot;, &quot;Xms&quot;: &quot;1024m&quot;}</code></td>
</tr>
</tbody>
</table>

4. Click **Add Parameter** and then commit changes.
5. Run Puppet on the appropriate nodes to apply the change. If you’re running it on the console node, the console will be unavailable briefly while `pe-console-services` restarts.

<table>
<thead>
<tr>
<th>Service</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe-console-services</td>
<td>console</td>
</tr>
<tr>
<td>Puppet Server</td>
<td>master and compile masters</td>
</tr>
<tr>
<td>orchestration-services</td>
<td>master</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>PuppetDB</td>
</tr>
</tbody>
</table>

**Related information**

Running Puppet on nodes on page 364

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.

**Increase ActiveMQ heap usage (master only)**

ActiveMQ uses a default heap size of 512MB, but you can increase this as needed.

512MB is the best value for mid-sized deployments, but can be a problem when building small proof-of-concept deployments on memory-starved VMs.

**Note:** Ensure that you have sufficient free memory before increasing the memory that is used by ActiveMQ. The increase shown below is only an example.

1. In the console, click **Nodes > Classification**, and in the **PE Infrastructure** node group, select the **PE ActiveMQ Broker** group.
2. Click **Configuration** and locate the `puppet_enterprise::profile::amq::broker` class.
3. Click the **Parameter name** list, select `heap_mb`, and in the **value** field, enter 1024.
4. Click **Add Parameter**, and commit changes.
5. On the command line on the master, run `puppet agent -t` to start a Puppet run and apply the change.
Disable Java garbage collection logging

Java garbage collection logs can be useful when diagnosing performance issues with JVM-based PE services. Garbage collection logs are enabled by default, and the results are captured in the support script, but you can disable them.

Disable garbage collection logging by editing your Hiera default `.yaml` file. Add any of the following parameters as needed:

```yaml
puppet_enterprise::console_services::enable_gc_logging: false
puppet_enterprise::master::puppetserver::enable_gc_logging: false
puppet_enterprise::profile::orchestrator::enable_gc_logging: false
puppet_enterprise::puppetdb::enable_gc_logging: false
puppet_enterprise::profile::amq::broker::enable_gc_logging: false
```

Configuring ulimit for PE services

As your infrastructure grows and you bring more agents under management, you may 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 will not be high enough 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-activemq`
- `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:

   ```sh
   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
<PE SERVICE USER> hard nofile 32768
```

---

**Tuning monolithic installations**

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. For example, you might need to allocate more memory per JRuby depending on the number of environments, or the number of agents and their run intervals.

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 compile masters.

**Related information**

- [Hardware requirements](#) on page 148
  These hardware requirements are based on internal testing at Puppet and are meant only as guidelines to help you determine your hardware needs.
- [Tune the maximum number of JRuby instances](#) on page 238
  The `jruby_max_active_instances` setting controls the maximum number of JRuby instances to allow on the Puppet Server.
- [Configure command processing threads](#) on page 245
  The `command_processing_threads` setting defines how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.
- [Increase the Java heap size for PE Java services](#) on page 251
  The Java heap size is the Java Virtual Machine (JVM) memory allocated to Java services in Puppet Enterprise (PE). You can use any configuration method you choose. We will use the console to change the Java heap size for console services, Puppet Server, orchestration services, or PuppetDB in the examples below.
- [Increase ActiveMQ heap usage (master only)](#) on page 252
  ActiveMQ uses a default heap size of 512MB, but you can increase this as needed.
- [Tuning the PostgreSQL buffer pool size](#) on page 244
If you are experiencing performance issues or instability with the console, adjust the buffer memory settings for PostgreSQL.

**Master tuning**

These are the default and recommended tuning settings for your master, master of masters (if you use compile masters), or high availability replica.

### Tuning for 4 cores, 8 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>PuppetDB</th>
<th>ActiveMQ</th>
<th>Orchestrator</th>
<th>PostgreSQL</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default 3</td>
<td>2048</td>
<td>512</td>
<td>2</td>
<td>256</td>
<td>512</td>
<td>256</td>
<td>192</td>
</tr>
<tr>
<td>Recommended</td>
<td>1024</td>
<td>512</td>
<td>1</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>2048</td>
</tr>
<tr>
<td>With 2 compile masters</td>
<td>2048</td>
<td>512</td>
<td>2</td>
<td>1024</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
</tbody>
</table>

### Tuning for 8 cores, 16 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>PuppetDB</th>
<th>ActiveMQ</th>
<th>Orchestrator</th>
<th>PostgreSQL</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default 4</td>
<td>2048</td>
<td>1024</td>
<td>4</td>
<td>256</td>
<td>512</td>
<td>256</td>
<td>192</td>
</tr>
<tr>
<td>Recommended</td>
<td>3840</td>
<td>1024</td>
<td>2</td>
<td>1024</td>
<td>768</td>
<td>768</td>
<td>4096</td>
</tr>
<tr>
<td>With 2 compile masters</td>
<td>1536</td>
<td>1024</td>
<td>4</td>
<td>3072</td>
<td>1024</td>
<td>768</td>
<td>768</td>
</tr>
</tbody>
</table>

### Tuning 16 cores, 32 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>PuppetDB</th>
<th>ActiveMQ</th>
<th>Orchestrator</th>
<th>PostgreSQL</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default 4</td>
<td>2048</td>
<td>2048</td>
<td>8</td>
<td>256</td>
<td>512</td>
<td>256</td>
<td>192</td>
</tr>
<tr>
<td>Recommended</td>
<td>11264</td>
<td>2048</td>
<td>4</td>
<td>2048</td>
<td>1024</td>
<td>1024</td>
<td>8192</td>
</tr>
<tr>
<td>With 4 compile masters</td>
<td>4096</td>
<td>2048</td>
<td>8</td>
<td>5120</td>
<td>1024</td>
<td>1024</td>
<td>8192</td>
</tr>
</tbody>
</table>
Compile master tuning

These are the default and recommended tuning settings for compile masters.

### Tuning 4 cores, 8 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JRuby max active instances</td>
<td>Java heap (MB)</td>
<td>Reserved code cache (MB)</td>
</tr>
<tr>
<td>Default</td>
<td>3</td>
<td>2048</td>
<td>512</td>
</tr>
<tr>
<td>Recommended</td>
<td>3</td>
<td>1536</td>
<td>512</td>
</tr>
</tbody>
</table>

### Tuning 8 cores, 16 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JRuby max active instances</td>
<td>Java heap (MB)</td>
<td>Reserved code cache (MB)</td>
</tr>
<tr>
<td>Default</td>
<td>4</td>
<td>2048</td>
<td>1024</td>
</tr>
<tr>
<td>Recommended</td>
<td>7</td>
<td>5376</td>
<td>1024</td>
</tr>
</tbody>
</table>

### Tuning 16 cores, 32 GB of RAM

<table>
<thead>
<tr>
<th>Install type</th>
<th>Puppet Server</th>
<th>CPU totals</th>
<th>Memory totals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JRuby max active instances</td>
<td>Java heap (MB)</td>
<td>Reserved code cache (MB)</td>
</tr>
<tr>
<td>Default</td>
<td>4</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>Recommended</td>
<td>15</td>
<td>15360</td>
<td>2048</td>
</tr>
</tbody>
</table>

### 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` on your master, it queries PuppetDB to identify infrastructure hosts and their processor and memory facts, and outputs settings in YAML format for use in Hiera.

The `puppet infrastructure tune` command optimizes based on available system resources, not agent load or environment complexity. You can add the option `--memory_per_jruby <MB>` to optimize the Puppet Server service for environment complexity.

With the `--current` option, you can review currently specified settings for PE services. Settings might be specified in either the console or in Hiera, with console settings taking precedence over Hiera settings. You should specify settings in the console or Hiera, but not both. The `--current` option identifies duplicate settings found in both places.

The `puppet infrastructure tune` command is compatible with monolithic and split infrastructures, with or without compile masters, external PostgreSQL hosts, and replica hosts. You can run the command on your master, but not on compile masters or high availability replicas. The command must be run as root.

For more information about the tune command, run `puppet infrastructure tune --help`.
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>
</table>
| { 
  "authorization": { 
    "version": 1  
  } 
} | "authorization": { 
  "version": 1  
} |

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>
</table>
| "authorization": { 
  "version": 1  
} | authorization: { 
  version: 1  
} |

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>
</table>
| Map  
rbac: { 
  password-reset-expiration: 24,  
  session-timeout: 60,  
  failed-attempts-lockout: 10,  
} | rbac: { 
  password-reset-expiration: 24  
  session-timeout: 60  
  failed-attempts-lockout: 10  
} |
| Array  
http-client: { 
  ssl-protocols: [TLSv1, TLSv1.1, TLSv1.2]  
} | http-client: { 
  ssl-protocols: [TLSv1, TLSv1.1, TLSv1.2]  
} |
Comments
Add comments using either // or #. Inline comments are supported.

HOCON example

```hocon
authorization: {
    version: 1
    rules: [
        {
            # Allow nodes to retrieve their own catalog
            match-request: {
                path: "^/puppet/v3/catalog/([^/]+)\$"
                type: regex
                method: [get, post]
            }
        }
    ]
}
```

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.

**Related information**

Methods for configuring Puppet Enterprise on page 235

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 own certificate to the whitelist, increase the max-threads setting for http and https requests, or configure the number of JRuby instances.

**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, master, and agent information:

- License UUID
- Number of licensed nodes
- Product name
- PE version
- Master's operating system
- Master's public IP address
- Whether the master is running on Microsoft Azure
- The hypervisor the master 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 high availability replicas, if applicable

Puppet Enterprise feature use information:

- Number of node groups in use
• Number of nodes used in orchestrator jobs since last orchestrator restart
• Mean nodes per orchestrator job
• Maximum nodes per orchestrator job
• Minimum nodes per orchestrator job
• Total orchestrator jobs created since last orchestrator restart
• Number of non-default user roles in use
• Whether MCollective is in use
• Type of certificate autosigning in use
• Number of nodes in the job that were run over Puppet Communications Protocol
• List of Puppet task jobs
• List of Puppet deploy jobs
• 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

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 master
• Time to update PE classification for new server
• Time to deactivate the old Master of Masters 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 backup 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
• 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

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

© 2020 Puppet, Inc.
The collected data is tied to a unique, anonymized identifier for each master and your site as a whole. No personally identifiable information is collected, and the data we collect is never used or shared outside Puppet, Inc.

**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 must use text-mode installation. Text-mode installation uses the `pe.conf` file to set parameters and values needed for the install.

If you prefer to use one of the non-text-mode installation methods, you can opt out of data collection after installation.

1. Follow the instructions for a monolithic text-mode install or a split text-mode install.
2. When you edit the `pe.conf` file, set the `puppet_enterprise::send_analytics_data` variable to `false`:

   "puppet_enterprise::send_analytics_data": false

3. Follow the rest of the text-mode installation instructions.

By running Puppet with this setting, you will be opted out of data collection.

**Opt out after installing**

If you've already installed PE and want to disable data collection, follow these steps.

1. In the console, click **Classification**, and then click **PE Infrastructure**.
2. On the **Configuration** tab, on the `puppet_enterprise` class, add `send_analytics_data` as a parameter
   and set the **Value** to `false`.
3. On the **Inventory** page, select your master node 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 in Puppet Enterprise**

A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources on a node that have `source` attributes. Using static catalogs can reduce the number of requests an agent makes to the master.

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A master typically compiles a catalog from manifests of Puppet code.

A static catalog includes metadata that specifies the desired state of any file resources on a node that have `source` attributes pointing to `puppet://` locations. This metadata can refer to a specific version of the file, rather than the latest version, and can confirm that the agent is applying the appropriate version of the file resource for the catalog. Also, because the metadata is provided in the catalog, agents make fewer requests to the master.

See the open source Puppet documentation more information about Resources, File types, and Catalog compilation.
Enabling static catalogs

When a master 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 will 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. In Puppet Enterprise (PE), such situations are particularly likely if you’ve configured your agents to run off cached catalogs for participation in application orchestration services.

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 master.

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 master each time the catalog's applied, even though nothing's changed in the cached catalog. This causes the master 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 master and requests a catalog.
• The master 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 master and requests a catalog.
• The master 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 master, 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. Once 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 will 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 master 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 will 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 621
File sync is normally enabled or disabled automatically along with Code Manager.

Running jobs with Puppet orchestrator on page 473
With the Puppet orchestrator, you can run two types of "jobs": on-demand Puppet runs or Puppet tasks.

Disabling static catalogs globally with Hiera

You can turn off all use of static catalogs with a Hiera setting.

1. Edit your Hiera default .yaml file.
2. Add or change this setting:

   puppet_enterprise::master::static_catalogs: false

Related information

Configure settings with Hiera on page 236
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.

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 Classification, and in the PE Infrastructure node group, select the PE Master node group.
3. On the Configuration tab, locate the puppet_enterprise::profile::master class, and select file_sync_enabled from its Parameter list.

© 2020 Puppet, Inc.
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 master.

Related information
Running Puppet on nodes on page 364
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.

Configuring high availability

Enabling high availability for Puppet Enterprise ensures that your system remains operational even if certain infrastructure components become unreachable.

- High availability on page 264
  High availability configuration creates a replica of your primary master.
- Configure high availability on page 269
  To configure high availability, you must provision and enable a replica to serve as backup during failovers. If your master is permanently disabled, you can then promote a replica.

High availability

High availability configuration creates a replica of your primary master.

The primary master is your Puppet master or, if your installation includes compile masters, your master of masters. The replica node is called the primary master replica. You may only have one provisioned replica at a time.

Note: We don’t support high availability for split installations — where the master, console, and PuppetDB components are installed on separate machines. Only monolithic installations — with or without compile masters and a master of masters — are supported.

There are two main advantages to enabling high availability:

- If your primary master fails, the replica takes over, continuing to perform critical operations.
- If your primary master can’t be repaired, you can promote the replica to master. Promotion establishes the replica as the new, permanent master.

Related information
What happens during failovers on page 266
Failover occurs when the replica takes over services usually performed by the master.

High availability architecture

The primary master replica is not an exact copy of the primary master. Rather, the replica duplicates specific infrastructure components and services. Hiera data and other custom configurations are not replicated.

Replication may be read-write, meaning that data can be written to the service or component on either the master or the replica, and the data is synced to both nodes. Alternatively, replication may be read-only, where data is written only to the master 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 compile master, redirecting PuppetDB and cert requests to the primary master.

<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>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>
</tbody>
</table>

**Important:** When you enable high availability, you must use or pe.conf only — not the console — to specify configuration parameters. Using pe.conf or ensures that configuration is applied to both your master and replica.

In a monolithic installation, when a Puppet run fails over, agents communicate with the replica instead of the master. In a monolithic installation with compile masters, agents communicate with load balancers or compile masters, which communicate with the master or replica.

Figure 1: Monolithic HA installation with compile masters

**Related information**

Configure settings with Hiera on page 236

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.
What happens during failovers

Failover occurs when the replica takes over services usually performed by the master.

Failover is automatic — you don’t have to take action to activate the replica. With high availability enabled, Puppet runs are directed first to the master. If the primary master is either fully or partially unreachable, runs are directed to the replica.

In partial failovers, Puppet runs may use the server, node classifier, or PuppetDB on the replica if those services aren’t reachable on the master. For example, if the master’s node classifier fails, but its Puppet Server is still running, agent runs use the Puppet Server on the master 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
- Application orchestration

System and software requirements

Your Puppet infrastructure must meet specific requirements in order to configure high availability.

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>All supported PE master platforms.</td>
</tr>
<tr>
<td></td>
<td>• You must use Code Manager so that code is deployed to both the master and the replica after you enable a replica.</td>
</tr>
<tr>
<td></td>
<td>• You must use the default PE node classifier so that high availability classification can be applied to nodes.</td>
</tr>
<tr>
<td></td>
<td>• Orchestrator must be enabled so that agents are updated when you provision or enable a replica. Orchestrator is enabled by default.</td>
</tr>
<tr>
<td>Replica</td>
<td>• 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.</td>
</tr>
<tr>
<td>Software</td>
<td>• Must have the same hardware specifications and capabilities as your primary master node.</td>
</tr>
</tbody>
</table>
### Component | Requirement
--- | ---
Firewall | Both the master and the replica must comply with these port requirements:  
- Firewall configuration for monolithic installations with compile masters. These requirements apply whether your HA environment uses a single master or compile masters.  
- Port 5432 must be open to facilitate database replication by console services.

Node names |  
- You must use resolvable domain names when specifying node names for the master 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**

**Managing and deploying Puppet code** on page 562  
Puppet Enterprise (PE) includes built-in tools for managing and deploying your Puppet infrastructure. Code Manager and r10k are code management tools that automatically install modules, create and maintain environments, and deploy new code to your masters, all based on version control of your Puppet code and data.

**Configure the orchestrator and pe-orchestration-services** on page 247  
There are several parameters you can add to configure the behavior of the orchestrator and pe-orchestration-services.

**Firewall configuration for monolithic installations with compile masters** on page 160  
These are the port requirements for monolithic installations with compile masters.

**Generate a token using puppet-access** on page 290  
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 Puppet master.

**Classification changes in high availability installations**

When you provision and enable a replica, the system makes a number of classification changes in order to manage high availability without affecting existing configuration.

These preconfigured node groups are added in high availability installations:

<table>
<thead>
<tr>
<th>Node group</th>
<th>Matching nodes</th>
<th>Inherits from</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE HA Master</td>
<td>primary master</td>
<td>PE Master node group</td>
</tr>
<tr>
<td>PE HA Replica</td>
<td>primary master replicas</td>
<td>PE Infrastructure node group</td>
</tr>
</tbody>
</table>

These parameters are used to configure high availability installations:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
<th>Node group</th>
<th>Classes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent-server-urls</td>
<td>Specifies the list of servers that agents contact, in order.</td>
<td>PE Agent, PE Infrastructure Agent</td>
<td>puppet_enterprise::profile::agent</td>
<td>In monolithic installations with compile masters, agents must be configured to communicate with the load balancers or compile masters. <strong>Important:</strong> Setting agents to communicate directly with the replica in order to use the replica as a compile master is not a supported configuration.</td>
</tr>
<tr>
<td>replication_mode</td>
<td>Sets replication type and direction on masters and replicas.</td>
<td>PE Master (none), HA Master (source), HA Replica (replica)</td>
<td>puppet_enterprise::profile::master, puppet_enterprise::profile::database, puppet_enterprise::profile::console</td>
<td>System parameter. Don't modify.</td>
</tr>
<tr>
<td>ha_enabled_replicas</td>
<td>Tracks replica nodes that are failover ready.</td>
<td>PE Infrastructure</td>
<td>puppet_enterprise::profile::agent</td>
<td>System parameter. Don't modify. Updated when you enable a replica.</td>
</tr>
<tr>
<td>pcp_broker_list</td>
<td>Specifies the list of Puppet Communications Protocol brokers that Puppet Execution Protocol agents contact, in order.</td>
<td>PE Agent, PE Infrastructure Agent</td>
<td>puppet_enterprise::profile::agent</td>
<td></td>
</tr>
</tbody>
</table>

**Load balancer timeout in high availability installations**

High availability 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 compile masters 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 high availability

To configure high availability, you must provision and enable a replica to serve as backup during failovers. If your master is permanently disabled, you can then promote a replica.

**Before you begin**

Apply high availability system and software requirements.

**Tip:** High availability is configured and managed with `puppet infrastructure` commands, many of which require a valid admin RBAC token. For details about these commands, on the command line, run `puppet infrastructure help <ACTION>`, for example, `puppet infrastructure help provision`.

**Related information**

- Generate a token using puppet-access on page 290
- System and software requirements on page 266

**Provision a replica**

Provisioning a replica duplicates specific components and services from the master to the replica.

**Before you begin**

Ensure you have a valid admin RBAC token. See Generate a token using puppet-access on page 290.

**Note:** While completing this task, the master is unavailable to serve catalog requests. Time completing this task accordingly.

1. Ensure that the node you're provisioning as a replica is set to use the primary master as its Puppet Server.
   
   On the prospective replica node, in the `/etc/puppetlabs/puppet/puppet.conf` file's main section, set the server variable to the node name of the primary master. For example:
   
   ```
   [main]
   certname = <REPLICA NODE NAME>
   server = <MASTER NODE NAME>
   ```

2. On the primary master, as the root user, run `puppet infrastructure provision replica <REPLICA NODE NAME>`

   After the provision command completes, services begin syncing from the master to the replica. The amount of time the sync takes depends on the size of your PuppetDB and the capability of your hardware. Typical installations take 10-30 minutes. With large data sets, you can optionally do a manual PuppetDB replication to speed installation.

   **Note:** All `puppet infrastructure` commands 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.

3. (Optional) Verify that all services running on the primary master are also running on the replica:
   
   a) From the primary master, 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.

When provisioning is complete, you must enable the replica to complete your HA configuration.
Manually copy PuppetDB to speed replication
For large PuppetDB installations, you can speed initial replication by manually copying the database from the master to the replica. If you have already started automatic provisioning, you can manually copy your PuppetDB at any time during sync.

The size of your PuppetDB correlates with the number of nodes and resources in your Puppet catalogs. To optionally examine the size of your database, on the PuppetDB PostgreSQL node, run `sudo -u pe-postgres /opt/puppetlabs/server/bin/psql -c \"\l+ "<DB_NAME>\"\"`.

**Note:** By default, `<DB_NAME>` is `pe-puppetdb`.

1. On the PuppetDB node, export the database:

   ```shell
   sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump --format=custom --compress=3 --file=<DUMP_OUTPUT> --dbname="<DB_NAME>"
   ```

2. On the PuppetDB node, transfer the output using your preferred tool, such as SCP:

   ```shell
   scp -r <DUMP_OUTPUT> <REMOTE_USER>@<REPLICA_HOST>:<REPLICA_DUMP_OUTPUT>
   ```

3. On the primary replica node, restore PuppetDB:

   ```shell
   sudo puppet resource service puppet ensure=stopped
   sudo puppet resource service pe-puppetdb ensure=stopped
   sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_restore --clean --jobs=<PROCESSOR_COUNT> --dbname="<DB_NAME>" <REPLICA_DUMP_OUTPUT>
   sudo puppet resource service puppet ensure=running
   sudo puppet resource service pe-puppetdb ensure=running
   sudo puppet agent -t
   ```

After manual export and restore, PuppetDB automatically updates the replica with any changes that occurred on the master in the meantime.

Enable a 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**
- Back up your classifier hierarchy, because enabling a replica alters classification.
- Ensure you have a valid admin RBAC token. See [Generate a token using puppet-access](#) on page 290.

**Note:** While completing this task, the master is unavailable to serve catalog requests. Time completing this task accordingly.

1. On the primary master, as the root user, run `puppet infrastructure enable replica <REPLICA_NODE_NAME>`, then follow the prompts to instruct Puppet how to configure your deployment.

2. Deploy updated configuration to nodes by running Puppet or waiting for the next scheduled Puppet run.

   **Note:** If you use the direct 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}'`

3. 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 master:
   a) Prevent the replica and a test node from contacting the master. For example, you might temporarily shut down the master 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 may take longer than normal when in failover mode.
   c) Reconnect the replica and test node.
4. On the replica, in the /etc/puppetlabs/puppet/puppet.conf file's main section, remove the entire line where the server variable is set: `server = <MASTER NODE NAME>`

When you enable HA, the `server` variable is superseded by the `server_list` variable. If you later promote a replica on which the server variable still exists and points to the old master, some commands might not function properly.

5. If you've specified any tuning parameters for your master using the console, move them to Hiera instead.

Using Hiera ensures configuration is applied to both your master and replica.

**Related information**

- **Back up your PE infrastructure** on page 758
  PE backup creates a copy of your Puppet infrastructure, including configuration, certificates, code, and PuppetDB.

- **Running Puppet on nodes** on page 364
  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.

- **Direct Puppet: a workflow for controlling change** on page 485
  The orchestrator—used alongside other PE tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

- **Configure settings with Hiera** on page 236
  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.

- **Classification changes in high availability installations** on page 267
  When you provision and enable a replica, the system makes a number of classification changes in order to manage high availability without affecting existing configuration.

- **Managing agent communication in geo-diverse 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 *geo-diverse installations*, with load balancers or compile masters in multiple locations, you must manually configure agent communication settings so that agents fail over to the appropriate load balancer or compile master.

  To skip automatically configuring which Puppet servers and PCP brokers agents communicate with, use the `--skip-agent-config` flag when you enable a replica, for example:

  ```
puppet infrastructure enable replica example.puppet.com --skip-agent-config
  ```

  To manually configure which load balancer or compile master 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 compile master 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`.

© 2020 Puppet, Inc.
Promote a replica

If your master can’t be restored, you can promote the replica to master to establish the replica as the new, permanent master.

1. Verify that the primary master is permanently offline.
   
   If the primary master comes back online during promotion, your agents can get confused trying to connect to two active masters.

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 master, such as PE client tool configurations, Code Manager hooks, Razor brokers, and CNAME records.

4. Deploy updated configuration to nodes by running Puppet or waiting for the next scheduled run.
   
   **Note:** If you use the direct 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}'`

5. (Optional) Provision a new replica in order to maintain high availability.
   
   **Note:** Agent configuration must be updated before provisioning a new replica. If you re-use your old master’s node name for the new replica, agents with outdated configuration might use the new replica as a master before it’s fully provisioned.

Related information

**Running Puppet on nodes** on page 364

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.

**Direct Puppet: a workflow for controlling change** on page 485

The orchestrator—used alongside other PE tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

Enable a new replica using a failed master

After promoting a replica, you can use your old master as a new replica, effectively swapping the roles of your failed master and promoted replica.

**Before you begin**

The `puppet infrastructure run` command leverages built-in plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your master 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 master. For more information, see [OpenSSH configuration options](#).

On your promoted replica logged in as root, run `puppet infrastructure run enable_ha_failover`, specifying these parameters:

- `host` — Hostname of the failed master. This node becomes your new replica.
- `caserver` — Hostname of the promoted replica that’s serving as your new master.

  **Note:** If you’re running PE version 2018.1.11 or newer, do not include the `caserver` parameter.

- `topology` — Architecture used in your environment, either `mono` or `mono-with-compile`
- `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:

```
puppet infrastructure run enable_ha_failover host=<FAILED_MASTER_HOSTNAME>
caserver=<PROMOTED_REPLICA_HOSTNAME> topology=mono
```

The failed master is repurposed as a new replica.

**Related information**

Generate a token using puppet-access on page 290

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 Puppet master.

**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. See Generate a token using puppet-access on page 290.

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. Verify that the replica to be removed is permanently offline.
2. On the primary master, as the root user, run `puppet infrastructure forget <REPLICA NODE NAME>`
   The replica is decommissioned, the node is purged as an agent, and a Puppet run is completed on the master.

**Reinitialize a replica**

If you encounter certain errors on your primary master replica after provisioning, you can reinitialize the replica. Reinitializing destroys and re-creates replica databases, as specified.

**Before you begin**

Your primary master must be fully functional and the replica must be able to communicate with the primary master.

⚠️ **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 primary master replica, 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`.
     **Important**: The replica must be running PE version 2018.1.8 or newer to reinitialize specific databases using the `--db` flag.
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

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.

Accepting the console's certificate

The console uses an SSL certificate created by your own local Puppet certificate authority. Since this authority is specific to your site, web browsers won't know it or trust it, and you'll have to add a security exception in order to access the console.

Adding a security exception for the console is safe to do. Your web browser will warn 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. Since 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.

Since the console is the main point of control for your infrastructure, you probably want to decline your browser's offer to remember its password.

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, click Access control > Users.
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 admin password

In RBAC, one of the built-in users is the admin, a superuser with all available read/write privileges. To reset the admin password for console access, run the `set_console_admin_password.rb` utility script.

1. Log into the console node as the root user.
   
   **Note:** You must run the script from the command line of the server installed with the console component. In a split install, it cannot be run from the Puppet master.

2. Run the `set_console_admin_password.rb` script:

   ```bash
   /opt/puppetlabs/puppet/bin/ruby /opt/puppetlabs/server/bin/
   set_console_admin_password.rb <NEW_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
```

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 276
- **Creating and managing local users and user roles** on page 281

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.
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 four default user roles: Administrators, Code Deployers, Operators, and Viewers. In addition, you can 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**

The table below lists the available object types and their permissions, as well as an explanation of the permission and how to use it.

Note that 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>Action (display name)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate requests</td>
<td>Accept and reject</td>
<td>Accept and reject certificate signing requests. Object must always be &quot;*&quot;.</td>
</tr>
<tr>
<td>Console</td>
<td>View</td>
<td>View the PE console. Object must always be &quot;*&quot;.</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Type</th>
<th>Action (display name)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</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>
</tr>
<tr>
<td>Puppet environment</td>
<td>Deploy code</td>
<td>Deploy code to a specific PE environment.</td>
</tr>
<tr>
<td>Scheduled jobs</td>
<td>Delete another user's scheduled jobs.</td>
<td>Delete jobs that another user has scheduled to run.</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>
</tr>
<tr>
<td>User groups</td>
<td>Delete</td>
<td>Delete a user group. This may be granted per group.</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>
</tr>
<tr>
<td>User roles</td>
<td>Create</td>
<td>Create new user roles. Object must always be &quot;*&quot;.</td>
</tr>
<tr>
<td>User roles</td>
<td>Edit</td>
<td>Edit and delete existing user roles. Object must always be &quot;*&quot;.</td>
</tr>
<tr>
<td>User roles</td>
<td>Edit members</td>
<td>Change which users and groups a role is assigned to. This may be granted per role.</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>
</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 may be granted per user.</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 may be granted per user.</td>
</tr>
<tr>
<td>Type</td>
<td>Action (display name)</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Users</td>
<td>Revoke</td>
<td>Revoke or disable a user account. 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 may be granted per user.</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>Action (display name)</th>
<th>Action (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>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>Scheduled jobs</td>
<td>scheduled_jobs</td>
<td>Delete another user’s scheduled jobs</td>
<td>delete</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>Type (display name)</td>
<td>Type (system name)</td>
<td>Action (display name)</td>
<td>Action (system name)</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>----------------------</td>
<td>----------------------</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 306

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;*&quot;&quot;)</td>
</tr>
<tr>
<td>Users</td>
<td>Edit</td>
<td>All (or &quot;*&quot;&quot;)</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. Only give directory_service:edit:* permission to users who are allowed see the password and other settings.
The ability to reset passwords should only be given 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 once the reset token is used. As such, the reset password permission should only be given 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.

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, click Access control > Users.
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 Users page, click 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, click Access control > User roles.
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 User roles page, click 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 will have permissions to view.
6. Click Add permission, and commit changes.

Related information
Best practices for assigning permissions on page 280
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 User roles page, click 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 will lose 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 User roles page, click 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, click Access control > Users.
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 Reinstate user access.

Delete a user
You can delete a user through the console. Note, however, that this action only deletes 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 will continue to be stored in the database and remain viewable through the API.
1. In the console, click **Access control > Users**.
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 will 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, click **Access control > User roles**.
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. This allows you to 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 your Active Directory or OpenLDAP 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 your external LDAP 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, click **Access control > External directory**.
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 will search the entire base DN for the user or group.

© 2020 Puppet, Inc.
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.example-service.net</td>
<td>myhost.delivery.example-service.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,dc=puppetlabs,dc=com</td>
<td>cn=admin,dc=delivery,dc=puppetlabs,dc=net</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>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 cert?</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 287.

**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` since both the group and the user are also under the organizational unit `ou=Puppet`.

**Example directory service entry**

```plaintext
# 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
```

© 2020 Puppet, Inc.
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 will log 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:

```
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 will be returned and the login will 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 only 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 only 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 only supports 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 311
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 Classification.
2. Open the PE Infrastructure node group and select the PE Console node group.
3. Click Configuration. 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 283. 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, under Access Control, select Users.
2. Select Administrator.
3. On the User details page, select 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, once 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, click Access control > User groups.

   **User groups** is only available 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, click Access control > User roles.
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.

© 2020 Puppet, Inc.
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 will only remove the group from Puppet Enterprise, not from the associated external directory service.

1. In the console, click **Access control > User groups**.

   **User groups** is only available 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.

It is important to remember that simply deleting a remote user’s PE account will 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

Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Authentication tokens manage access to the following PE services:

- Puppet orchestrator
- Code Manager
- Node classifier
- Role-based access control (RBAC)
- Activity service
- PuppetDB

You can generate tokens with the `puppet-access` command or with the API endpoint.

**Remember:** For security reasons, authentication tokens can only be generated for revocable users. The admin user and api_user cannot be revoked.

**Related information**

- Installing PE client tools on page 213

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily managed by Puppet.

- User permissions on page 277
The table below lists the available object types and their permissions, as well as an explanation of the permission and how to use it.

**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 Puppet master.

**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 five-minute lifetime, run `puppet-access login`.
   - To generate a token with a specific lifetime, run `puppet-access login --lifetime <TIME PERIOD>.

   For example, `puppet-access login --lifetime 5h` will generate 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.

**Related information**

*Setting a token-specific lifetime on page 293*

Tokens have a default lifetime of five minutes, 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**

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 `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, you may want to specify a longer token lifetime so that you don't have to regenerate the token too frequently.

**Configuring 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 Puppet master.

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 work station, you'll 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:

```
{
  "service-url": "https://<CONSOLE HOSTNAME>:4433/rbac-api",
  "token-file": "~//.puppetlabs/token",
  "certificate-file": "/etc/puppetlabs/puppet/ssl/certs/ca.pem"
}
```

PE will determine and add 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 will 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</td>
<td>-t, --token-file</td>
</tr>
<tr>
<td></td>
<td>~/puppetlabs/token.</td>
<td></td>
</tr>
<tr>
<td>certificate-file</td>
<td>The location of the CA that signed the console-services server's</td>
<td>--ca-cert</td>
</tr>
<tr>
<td></td>
<td>certificate. Defaults to the PE CA cert location, /etc/puppetlabs/puppet/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssl/certs/ca.pem.</td>
<td></td>
</tr>
<tr>
<td>config-file</td>
<td>Changes the location of your configuration file. Defaults to</td>
<td>-c, --config-file</td>
</tr>
<tr>
<td></td>
<td>~/puppetlabs/client-tools/puppet-access.conf.</td>
<td></td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Command line flag</th>
</tr>
</thead>
<tbody>
<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 only need to change this if you are moving the console server.</td>
<td>--service-url</td>
</tr>
</tbody>
</table>

### 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.

   ```bash
   curl -k -X POST -H 'Content-Type: application/json' -d '{"login": "<YOUR PE USER NAME>", "password": "<YOUR PE PASSWORD>"}' https://$<HOSTNAME>:4433/rbac-api/v1/auth/token
   ``

   The the command will return a JSON object containing the key token and the token itself.

   **CAUTION:** If you are using curl commands with the -k insecure SSL connection option, 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.

### 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'll use the /users/current endpoint of the RBAC v1 API to get information about the current authenticated user.

**Before you begin**

Generate a token and save it as an environment variable using `export TOKEN=<PASTE THE TOKEN HERE>`.

Run the following command:

   ```bash
   ``

   The command above uses the X-Authentication header to supply the token information.

   In some cases, such as GitHub webhooks, you may need to supply the token in a token parameter by specifying the request as follows:

   ```bash
   ``

   **CAUTION:** If you are using curl commands with the -k insecure SSL connection option, keep in mind that you are vulnerable to a person-in-the-middle attack.

### Change the token's default lifetime

Tokens have a default authentication lifetime of five minutes, 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 **Classification**.
2. Open the **PE Infrastructure** node group and click the **PE Console** node group.
3. On the Configuration 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 five minutes, 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"}.

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.

{"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.

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 may be viewed on the user’s Activity tab in the console.

You can revoke your own token by username, label, or full token, and may 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.

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, since 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.
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. Once 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>.

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, click Access control > Users. Click the full name of the user you are interested in.
2. Click the Activity tab.

RBAC API v1

The role-based access control (RBAC) service enables you to manage users, directory groups, and roles.

- **Endpoints** on page 295
  The role-based access control (RBAC) service enables you to manage users, directory groups, and roles.
- **Forming RBAC API requests** on page 295
  Web session authentication is required to access the RBAC API. You can authenticate requests by using either user authentication tokens or whitelisted certificates.
- **Users endpoints** on page 296
  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 301
  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 304
  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 306
  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 307
  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 309
  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 312
  When local users forget passwords or lock themselves out of PE by attempting to log in with incorrect credentials too many times, you'll have to 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.
• **RBAC service errors** on page 313
  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.
• **Configuration options** on page 316
  There are various configuration options for the RBAC service. Each section can exist in its own file or in separate files.

## Endpoints

The role-based access control (RBAC) service enables you to manage users, directory groups, and roles.

For general information about forming HTTP requests to the API, see the forming requests page. For information on errors encountered while using the RBAC v1 API, see the RBAC service errors page.

**Tip:** In addition to the endpoints on this page and in the v2 RBAC service API, there are endpoints that you can use to check the health of the RBAC service. These are available through the status API documentation.

<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>token</td>
<td>Generate the authentication tokens used to access PE.</td>
</tr>
<tr>
<td>rbac-service</td>
<td>Check the status of the RBAC service.</td>
</tr>
</tbody>
</table>

## Forming RBAC API requests

Web session authentication is required to access the RBAC API. You can authenticate requests by using either user authentication tokens or whitelisted 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.

### Authentication 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
```
The example above uses the X-Authentication header to supply the token information. In some cases, such as GitHub webhooks, you may need to supply the token in a token parameter. To supply the token in a token parameter, specify the request as follows:

```
```

⚠️ **CAUTION:** Be aware when using the token parameter method that the token parameter may be recorded in server access logs.

### Authentication using whitelisted certificate

You can also authenticate requests using a certificate listed in RBAC's certificate whitelist, located at `/etc/puppetlabs/console-services/rbac-certificate-whitelist`. 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 whitelisted certificate name (which must match a name in the `/etc/puppetlabs/console-services/rbac-certificate-whitelist` file) and the private key to run the script.

```
curl -X GET https://<HOSTNAME>:<PORT>/rbac-api/v1/users \
   --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem \
   --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem \
   --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem -H "Content-Type: application/json"
```

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 289
- **Content-type headers in the RBAC API**

RBAC only accepts 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`.

### 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

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. Must be unique among users and groups.</td>
<td>&quot;admin&quot;</td>
</tr>
<tr>
<td>email</td>
<td>An email address string. Not currently utilized by any code in PE.</td>
<td>&quot;<a href="mailto:hill@example.com">hill@example.com</a>&quot;</td>
</tr>
<tr>
<td>Key</td>
<td>Explanation</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</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. is_group should always be false for a user.</td>
<td>true/false</td>
</tr>
<tr>
<td>is_remote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_superuser</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_revoked</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>last_login</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>inherited_role_ids</td>
<td>(remote users only) An array of role IDs indicating which roles a remote user inherits from their groups.</td>
<td>[9 1 3]</td>
</tr>
<tr>
<td>group_ids</td>
<td>(remote users only) An array of UUIDs indicating which groups a remote user inherits roles from.</td>
<td>[&quot;3a96d280-54c9-11e4-916c-0800200c9a66&quot;]</td>
</tr>
</tbody>
</table>

**GET /users**

Fetches 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:

{"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": "2014-05-04T02:32:00Z"}

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": "2014-05-04T02:32:00Z"}

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:

```json
{"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 will be returned.

Error responses
If the email or login for the user conflicts with an existing user's login, a 409 Conflict response will be 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_id 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:

```json
{"id": "c8b2c380-5889-11e4-8ed6-0800200c9a66",
 **"login": "Amari",
 **"email": "amari@puppet.com",
 **"display_name": "Amari Perez",
 **"role_ids": [1, 2, 3]}
```
An example for a remote user:

```json
{"id": "3271fde0-588a-11e4-8ed6-0800200c9a66",
 "login": "Jean",
 "email": "jeanjackson@example.com",
 "display_name": "Jean Jackson",
 **"role_ids": [4, 1],**
 "inherited_role_ids": [],
 "group_ids": [],
 "is_group": false,
 "is_remote": true,
 **"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 will be 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:

```bash
curl -X DELETE \
   --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
   --cert /opt/puppet/share/puppet-dashboard/certs/pe-internal-dashboard.cert.pem \
   --key /opt/puppet/share/puppet-dashboard/certs/pe-internal-dashboard.private_key.pem \
   https://$(hostname -f):4433/rbac-api/v1/users/3982a629-1278-4e38-883e-12a7cac91535
```

**Response format**

If the user is successfully deleted, a 204 No Content response with an empty body will be returned.

**Error responses**

If the current user does not have the `users:edit` permission for this user group, a 403 Forbidden response will be returned.

If a user with the provided identifier does not exist, a 404 Not Found response will be 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

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>
<tr>
<td>is_revoked</td>
<td>Setting this flag to true currently does nothing for a group.</td>
<td>true/false</td>
</tr>
<tr>
<td>user_ids</td>
<td>An array of UUIDs indicating which users inherit roles from this group.</td>
<td>[&quot;3a96d280-54c9-11e4-916c-0800200c9a66&quot;]</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","5c1ab4b0-588b-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:

{"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 will be returned.

If the current user does not have the appropriate user permissions to request the group data, a 403 Forbidden response will be 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:

```
{"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 will be returned.

Error responses
If the login for the group conflicts with an existing group login, a 409 Conflict response will be 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:

```
{"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 will be 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 will be returned.

Error responses
If the current user does not have the user_groups:delete permission for this user group, a 403 Forbidden response will be returned.
If a group with the provided identifier does not exist, a 404 Not Found response will be 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

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>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>&quot;fc115750-555a-11e4-916c-0800200c9a66&quot;</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": [
    "fc115750-555a-11e4-916c-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": "*",
}]
```
POST /roles
Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

Request format
Accepts a new role object. Any of the arrays can be empty and "description" can be null.

For example:

```json
{"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.

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:

```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"}
```

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 will immediately lose the role and all permissions granted by it, but their session will be otherwise unaffected. Access to the next request that the user makes will be 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
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>
<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.

```
{
"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]
```

**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.

**Related information**

[Token-based authentication](#) on page 289
Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. 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 will be 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 will be 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 a whitelisted 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 may 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:

curl -cacert $(puppet config print cacert) -X POST -H 'Content-Type: application/json' -d '{"login": "<LOGIN>", "password": "<PASSWORD>", "lifetime": "4h", "label": "four-hour token"}' https://<HOSTNAME>:4433/rbac-api/v1/auth/token

The various parts of this curl command request are explained as follows:

• `-cacert [FILE]`: Specifies a CA certificate as described in Forming requests for the node classifier. Alternatively, you could use the `-k` 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 `-k` 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.

• `-X POST`: This is an HTTP POST request to provide your login information to the RBAC service.

• `-H '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.

• `-d '{"login": "YOUR PE USER NAME", "password": "<PASSWORD>", "lifetime": "4m", "label": "four-minute 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.

• `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.

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-w0duwdhjashd,kjsahdasoi0d9hw0hduashd0a9wdy0whdkaudhaksdhc9chakdh92..."}

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.

Related information

External directory settings on page 284
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:

```json
{"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:

```json
{}
```

**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:

```
{"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:

```
{"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 very 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 only searches 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 will be 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 may 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",
```
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’ll have to 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 admin password on page 275

In RBAC, one of the built-in users is the admin, a superuser with all available read/write privileges. To reset the admin password for console access, run the set_console_admin_password.rb utility script.

POST /users/:sid/password/reset

Generates a single-use password reset token for the specified local user.

The generated token can only be used to reset the password once and it has a limited lifetime. The lifetime is based on a configuration value 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:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhcGlfdXNlciIsInN1YiI6ImZyA4MTY3LWE1MDU0OGQzY2E1NjIiLCJlbWFpbCI6Indvbkx1Y1NlciIsImV4cCI6MTU0ODg4NTU0OCwiaWF0IjoxNTQ4Nzk5MTQ4fQ.qOG0w5UuZiBrT8yPoTG3YHwOhhkchrUh7kvFNiBFw0gPyfyZ2yhfjHcffJ-p0aUcdmBtuD95J9NGEAPtTwsDcuX0_33hPz03m0Rp3hDSjbiQwyTBFuf2rpqymaYo1VRw-Ecu60dulqBAkgQ9rmtotHtO6YThQ8M6p9RxIFk_u5YjZDrhLZ5qYxg-LY-cY21Fow_XFIVwc9vWBuDX11wLwLB9tQJoI2NVNBymoHABz0KrnV-dKGFrawF-gTvdtaVSoDFSxikkWkangGq2jb2Ghszhd8jdK9wWQ_8fn0cuk_kLlibE7Wr0htK045tT9BPazwZawHiojYFYyJfBiB_dF3quRE-4C2EbejQLd_u-WZGnJp2V_HGK4Spd56V-anuDq1aakaW31zDKJJuPI4CxE8_xpfFFLRdofTHrLPowo5dszbbK-9xw3JfIMhnsK4e2H_5Nywk0w951vonKuy...XFh2kn8zVTuKT19EyIPoPHcJwyA3Y569UXgkVUB9LAcwyEnEbU70kKXwlXZUfsQOzMIymkIR4kVqBxDgwnrABMEcTFK24_rkg8srKWP6f_TcEg3588C3_DTR8g

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:

```
{"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:

```
{"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 will include a message that specifies the cause of the failure.

**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 details 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 details key consists of one field, error, which contains the error message from the JSON parser.</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 msg contains a description of the problem. The details are an object with three keys:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• submitted: The submitted data as it was seen during schema validation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• schema: The expected structure of the data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• error: 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 id field with a different value. The details key consists of two fields, url-id and body-id, 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 api_user with a password (api_user 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. 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 <code>user_lookup_attr</code> 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>403</td>
<td></td>
</tr>
<tr>
<td>default-roles-immutable</td>
<td>403</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, <code>user_ids</code>) 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, <code>user_ids</code>) 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>422</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 <code>user_lookup_attr</code> 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>
Configuration options

There are various configuration options for the RBAC service. Each section can exist in its own file or in separate
files.

RBAC service configuration
You can configure the RBAC service's settings to specify the duration before inactive user accounts expire, adjust the
length of user sessions, the number of times a user can attempt to log in, and the length of time a password reset token
is valid. You can also define a whitelist of certificates.

These configuration parameters are not required, but when present must be under the rbac section, as in the
following example:

```json
rbac: {
    # Duration in days before an inactive account expires
    account-expiry-days: 1
    # Duration in minutes that idle user accounts are checked
    account-expiry-check-minutes: 60
    # Duration in hours that a password reset token is viable
    password-reset-expiration: 24
    # Duration in minutes that a session is viable
    session-timeout: 60
    failed-attempts-lockout: 10
}
```

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. To activate this 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.

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.

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.

password-reset-expiration
When a user doesn't remember their current password, an administrator can generate a token for them to change their
password. The duration, in hours, that this generated token is valid can be changed with the password-reset-
expiration config parameter. The default value is 24.

session-timeout
This parameter is a positive integer that specifies how long a user's session should last, in minutes. This session is the
same across node classifier, RBAC, and the console. The default value is 60.

failed-attempts-lockout
This parameter is a positive integer that specifies how many failed login attempts are allowed on an account before
that account is revoked. The default value is 10.

Note: If you change this value, create a new file or will Puppet reset back to 10 when it next runs. Create the file in
an RBAC section of /etc/puppetlabs/console-services/conf.d.
**certificate-whitelist**

This parameter is a path for specifying the file that contains the names of hosts that are allowed to use RBAC APIs and other downstream component APIs, such as the Node Classifier and the Activity services. This configuration is for the users who want to script interaction with the RBAC service.

Users must connect to the RBAC service with a client certificate that has been specified in this certificate-whitelist file. A successful match of the client certificate and this certificate-whitelist allows access to the RBAC APIs as the api_user. By default, this user is an administrator and has all available permissions.

The certificate whitelist contains, at minimum, the certificate for the nodes PE is installed on (one certificate for a monolithic install, and three certificates for a split install).

**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 options 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.
RBAC API v2

The role-based access control (RBAC) service enables you to manage users, directory groups, and roles.

- Tokens endpoints on page 318

A user’s access to PE services can be controlled using authentication tokens. Users can revoke authentication tokens using the tokens endpoints.

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 289

Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. 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 may 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": [
        "eyJhbGciOiJSUzUxM1IiXVc1NdeCI6MTQzOTQ5Mzg0NiwiZXhwIi0iODfSxf91351E7vi97s-9fFchHFpzbQzY60GDyRVJvsohTwpJNM1Ax0UBitgqCXYnE4rr8wKBoeOXoeEqezJ2u8-q0TgeN3keazDxdIqfhZ7H10-jDR0C5yeSBGWfX-0KEbp42cGz81A6rrIHpSaaajRcWUg9yTheUkT2crh6878orCLgfoBLdh-rOBTLelual6sash-
ggpdHqVkJFOoomEXM6UTJ1p1NuP01rNjM1xWhl8WpEpExH11_-136D1NM32kwo-oV6GzXRx70xq_N2CwIwObw-X15saUC4KkyPtDmNnvvC1n4"
    ],
    "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>400</td>
<td>Malformed, Sent if one of 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 revoke_tokens, revoke_tokens_by_usernames, nor revoke_tokens_by_labels is supplied.</td>
</tr>
<tr>
<td></td>
<td>• There are unrecognized query parameters or fields in the request body, 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_labels": [],
 "malformed_usernames": [],
 "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}.

Currently, this API may only be used 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.

```json
{"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}}
```
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 321
  Web session authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or whitelisted certificates.
- **Event types reported by the activity service** on page 322
  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 324
  Use the events endpoints to retrieve activity service events.

Forming activity service API requests

Web session authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or whitelisted 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. 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
   curl -k -X GET https://<HOSTNAME>:<PORT>/activity-api/v1/events?
   service_id=classifier -H "X-Authentication:$TOKEN"
   ```

Authentication using whitelisted certificate

You can also authenticate requests using a certificate listed in RBAC's certificate whitelist, located at /etc/puppetlabs/console-services/rbac-certificate-whitelist. 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 whitelisted certificate name (which must match a name in the /etc/puppetlabs/console-services/rbac-certificate-whitelist file) and the private key to run the script.

```bash
curl -X GET https://<HOSTNAME>:<PORT>/activity-api/v1/events?
service_id=classifier
   --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem
   --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem
   --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem
```

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 289
Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. 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 reported.</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 the affected user are displayed.</td>
<td>User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba) added to role Operators.</td>
</tr>
<tr>
<td>Authentication</td>
<td>A user logs in. The display name and user ID of the affected user are displayed.</td>
<td>User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba) logged in.</td>
</tr>
<tr>
<td>Password reset token</td>
<td>A token is generated for a user to use when resetting their password. The display name and user ID of the affected user are shown.</td>
<td>A password reset token was generated for user Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba).</td>
</tr>
<tr>
<td>Password changed</td>
<td>A user successfully changes their password with a token.</td>
<td>Password reset for user Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba).</td>
</tr>
<tr>
<td>Revocation</td>
<td>A user is revoked or reinstated.</td>
<td>User revoked.</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-aa15-123b93f75c7ba) 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>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-aa15-123b93f75c7ba) 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-aa15-123b93f75c7ba 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-e976f7c7f4bd) 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>
<tr>
<td>Revocation by username</td>
<td>All tokens for a username are revoked. These events are exposed in the console on the Activity tab for the user performing the revocation.</td>
<td>Administrator (42bf351c-f9ec-40af-84ad-e976f7c7f4bd) revoked all authentication tokens belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c).</td>
</tr>
</tbody>
</table>

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.

- `service_id`
  - **required** The ID of the service: classifier, code-manager, pe-console, or rbac.

- `subject_type`
  - The subject who performed the activity: users.

- `subject_id`
  - Comma-separated list of IDs associated with the defined subject type.
**object_type**

The object affected by the activity: node, node_groups, users, user_groups, roles, environment, or directory_server_settings.

**object_id**

Comma-separated list of IDs associated with the defined object type.

**offset**

Number of commits to skip before returning events.

**limit**

Maximum number of events to return. Default is 1000 events.

**Examples**

curl -k -X GET "https://web5.mydomain.edu:4433/activity-api/v1/events?service_id=classifier&subject_type=users&subject_id=6868e4af-2996-46c6-8e42-1ae873f8a0ba" -H "X-Authentication:$TOKEN"


**Response format**

Responses are returned in a structured JSON format. Example return:

```json
{
"commits": [{
  "object": {
    "id": "b55c209d-e68f-4096-9a2c-5ae52dd2500c", "name": "web_servers"
  },
  "subject": {
    "id": "6868e4af-2996-46c6-8e42-1ae873f8a0ba", "name": "kai.evens"
  },
  "timestamp": "2019-03-05T18:52:27Z", "events": [ {
    "message": "Deleted the \"web_servers\" group with id b55c209d-e68f-4096-9a2c-5ae52dd2500c"
  }]
}, "offset": 0, "limit": 1, "total-rows": 5
}

{
"commits": [{
  "object": {
    "id": "emerald-1.platform9.mydomain.edu", "name": "emerald-1.platform9.mydomain.edu"
  },
  "subject": {
    "id": "d1f4919a-cf65-4c26-8832-8c4b5302b58b", "name": "steve"
  },
  "timestamp": "2019-05-02T22:50:16Z", "events": [ {
    "message": "Scheduled job 3 request for facter_task task on emerald-1.platform9.mydomain.edu run via orchestrator job 7,759"
  }]
}, "offset": 0, "limit": 1, "total-rows": 2915
```
GET /v1/events.csv
Fetched activity service events and returns in a flat CSV format. Web session authentication is required.

Request format
The /v1/events.csv endpoint supports the same parameters as /v1/events

Examples

curl -k -X GET "https://web5.mydomain.edu:4433/activity-api/v1/events.csv?service_id=classifier&subject_type=users&subject_id=6868e4af-2996-46c6-8e42-1ae873f8a0ba -H "X-Authentication:$TOKEN"

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, "Changes 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, "Changes 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, "Changes 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, "Changes the parent to ec519937-8681-43d3-8b74-380d65736dba"
YYYY-MM-DD
00:41:18.944, users, 6868e4af-2996-46c6-8e42-1ae873f8a0ba, kai.evens, node_groups, ec519937-8681-43d3-8b74-380d65736dba, PE Orchestrator, edit, node_group_class_parameter, delete_node_group_class_parameter_puppet_enterprise::profile::orchestrator_use_application_services, "Removes the "use_application_services" parameter from the "puppet_enterprise::profile::orchestrator" class"
YYYY-MM-DD
00:41:10.631, users, 6868e4af-2996-46c6-8e42-1ae873f8a0ba, kai.evens, node_groups, ec519937-8681-43d3-8b74-380d65736dba, PE Orchestrator, edit, node_group_class_parameter, add_node_group_class_parameter_puppet_enterprise::profile::orchestrator_use_application_services, "Add the "use_application_services" parameter to the "puppet_enterprise::profile::orchestrator" class"
YYYY-MM-DD
20:41:30.223, users, 6868e4af-2996-46c6-8e42-1ae873f8a0ba, kai.evens, node_groups, 46e34005-bc48-48132212e9fbb, PE Agent, schedule_deploy, node_group, schedule_puppet_agent_on_node_group, Schedule puppet agent run on nodes in this group to be run at 2019-01-16T08:00:00Z
Inspection your infrastructure

The Puppet Enterprise 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, and investigate problems. These tools are grouped in the Inspect section of the console's sidebar.

- Monitoring current infrastructure state on page 327
  When nodes fetch their configurations from the Puppet master, they send back inventory data and a report of their run. This information is summarized on the Overview page in the console.

- Exploring your catalog with the node graph on page 332
  The node graph provides a graphic representation of any node's configuration, shows you the relationships between classes and resources, and lets you gain greater insight into just how the Puppet master compiles your code and ships it to an agent. Visualizing relationships between resources helps you organize them, creating more reliable deployments.

- Viewing and managing all packages in use on page 333
  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.

- Infrastructure reports on page 335
  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 338
  On the Events page in the console you'll find 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.

- Viewing and managing Puppet Server metrics on page 340
  Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

- Status API on page 350
  The status API allows you to check the health of PE components. It can be useful in automated monitoring of your PE infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or when troubleshooting problems in PE.

Monitoring current infrastructure state

When nodes fetch their configurations from the Puppet master, they send back inventory data and a report of their run. This information is summarized on the Overview page in the console.

The Overview 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 Puppet master is managing, and whether any nodes are unresponsive.

Node run statuses

The Overview 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 Puppet master, 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 only available on agent nodes running PE 2016.4 and later. Agents running earlier versions will 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 only available on agent nodes running PE 2016.4 and later. Agents running earlier versions will 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 Puppet master recently. Something might be wrong. The cutoff for considering a node unresponsive defaults to one hour, and can be configured.

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 Overview 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 *Overview* 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>!</td>
<td>Failure</td>
<td>Corrective change, intentional change, no change</td>
</tr>
<tr>
<td>!</td>
<td>Corrective change</td>
<td>Intentional change, no change</td>
</tr>
<tr>
<td>!</td>
<td>Intentional change</td>
<td>No change</td>
</tr>
<tr>
<td>✓</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 Overview page
You can filter the list of nodes displayed on the Overview 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 Overview 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 Overview 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>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 will display 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 may 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>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 )</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: )</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 )</td>
<td>inventory[certname] { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }</td>
</tr>
</tbody>
</table>

Monitor PE services

Puppet Enterprise includes tools for monitoring the status of core services including the activity, classifier, and RBAC services, Puppet Server, and PuppetDB. You can monitor these services on the command line, and from within the console.

View the Puppet Services status monitor

The Puppet Services status monitor provides a visual overview of the current state of key services, and can be used to quickly determine whether an unresponsive or restarting service is causing an issue with your deployment.

1. In the console, click Overview.
2. Click **Puppet Services status** to open the monitor.

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.

**puppet infrastructure status command**

The `puppet infrastructure status` command displays errors and alerts from PE services, including the activity, classifier, and RBAC services, Puppet Server, and PuppetDB.

If high availability is enabled in your environment, the command reports separately on the primary master, primary master replica, and compile masters. The command must be run as root.

### Exploring your catalog with the node graph

The node graph provides a graphic representation of any node's configuration, shows you the relationships between classes and resources, and lets you gain greater insight into just how the Puppet master compiles your code and ships it to an agent. Visualizing relationships between resources helps you organize them, creating more reliable deployments.

### How the node graph can help you

The node graph displays a node’s catalog (as of the last Puppet run) as an interactive visual map. The graph shows the desired state for each resource that PE manages, as well as each resource's status as of the last run, and helps you understand the dependencies between resources. It also allows you to visually identify complexity you may not need, and problematic dependencies that need your attention.

The node graph is an ideal tool for:

- Gaining greater insight into your deployment.
- Visualizing the relationship of resources on your nodes.
- Diagnosing dependency loops and viewing all impacted resources.
- Helping new users to understand relationships among classes and resources in their Puppet catalog.
- Understanding defined resource types as they are deployed across your configuration.
- Understanding the content of modules.

### Investigate a change with the node graph

Use the node graph to view the details of a change made to one of your nodes, and see how other resources and classes might be impacted.

**Access a node's graph**

You can reach a node's graph from either the **Overview** page or the node's detail page. If you're using the node graph to find the cause of a particular run status, use the **Overview** page path. If you already know which node you're interested in, the **Nodes** page path is most efficient.

1. Locate a node you're interested in viewing:
   - On the **Overview** page, filter by run status or fact value to locate the node.
   - On the **Nodes** page, search for the node by name.

2. Find and click the **Node graph** link:
   - On the **Overview** page, the **Node graph** link is located to the right of the node's name.
   - On the **Nodes** page, click the node name. The **View node graph** link is located at the top of the node details page.
Focus on a change
When the node graph opens, you’ll see the containment view, which presents the catalog in its entirety. This can be a bit overwhelming, but you can use filters to quickly focus on the change event you’re interested in.

1. In the Filter: bar, select status.
2. Select a run status, such as Corrective change.
   
   The Node graph details pane at the right of the screen opens and displays a list of resources that match your filter criteria.
3. Select any item on the list, and the graph repositions to focus on that resource.
   
   The Node graph details pane now shows additional data on current event status, as well as the object’s source file, code line number, relevant tags, and class containment information.

Investigate dependencies
If your resource is part of a dependency chain, a Dependency view link is shown in the Node graph details pane. This link does not appear for resources without any dependencies.

1. Click Dependency view.
   
   The Node graph details pane displays a list of the resource’s ancestor and descendant dependencies.
2. Hover over any item on the list, and the graph highlights that dependency path.
   
   Note: In the event that a dependency cycle is detected, the node graph issues a cycle warning message.
3. Click Exit dependency view to return to the containment view.

Viewing and managing all packages in use
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 inventory reporting is available for all nodes with Puppet agent version 1.6.0 or later installed, including systems that are not actively managed by Puppet.

Tip: Packages are gathered from all available providers. The package data reported on the Packages page can also be obtained by using the puppet resource command to search for package.

Enable package data collection
Package data collection is disabled by default, so the Packages page in the console will initially appear 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. Any node with Puppet agent version 1.6.0 or later installed can report package data, including nodes that are not under active management with Puppet Enterprise.

1. In the console, click Classification.
   
   • 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. Once the new node group is set up, use the Rules tab to dynamically add the relevant nodes.
2. Click Configuration. 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 will enable package inventory collection on this Puppet run, and will begin 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.

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 Classification 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 will look 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 Classification and select the node group you used when enabling package data collection.
2. Click Configuration.
3. In the puppet Enterprise::Profile::Agent class, locate package_inventory_enabled and click Edit.
4. Change the **Value of package_inventory_enabled** to **false**, then commit changes.

5. Run Puppet to apply these changes to the nodes in your node group and disable package data collection.
   Package data will be collected for a final time during this run.

6. Run Puppet a second time to purge package data from the impacted nodes' storage.

---

**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>Correction applied</th>
<th>Number of resources that received a corrective change after Puppet identified resources that were out of sync with the applied catalog.</th>
</tr>
</thead>
<tbody>
<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. 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.</td>
</tr>
</tbody>
</table>

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).

**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.
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 will display 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 may match any rule.
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 Puppet master that submitted the report to PuppetDB.</td>
</tr>
<tr>
<td>Puppet environment</td>
<td>The environment assigned to the node.</td>
</tr>
</tbody>
</table>
| Puppet run            | • The time that the Puppet run began  
                         • The time that the Puppet master 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 |
Analyzing changes across Puppet runs

On the Events page in the console you'll find 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 Puppet master at the end of the run. These reports provide the data presented on the Events page in the console.

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 --noop command-line option, the noop setting, or the noop =&gt; true metaparameter. Instead of making a corrective change, Puppet logs a corrective no-op event and reports the change it would have made.</td>
</tr>
</tbody>
</table>
## Intentional no-op

Puppet would have applied catalog changes to a property, but Puppet was instructed to not make changes on this resource, via either the `--noop` command-line option, the `noop` setting, or the `noop => true` metaparameter. Instead of making an intentional change, Puppet logs an intentional no-op event and reports the change it would have made.

## Skip

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 `schedule` metaparameter. The resource might be in sync or out of sync; Puppet doesn't know yet.

If the `schedule` 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 `schedule`, but does not apply to built-in scheduled tasks that happen weekly, daily, or at other intervals.

### 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 will report 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 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 since 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 puppetdb class causes the pe-puppetdb 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.

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 once. 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.

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 track Puppet Server metrics by using:

- Customizable, networked Graphite and Grafana instances
- A built-in experimental developer dashboard
- Status API endpoints

Note: None of these methods are officially supported. The Grafanadash and Puppet-graphite modules referenced in this document are not Puppet-supported modules; they are mentioned for testing and demonstration purposes only. The developer dashboard is a tech preview. Both the Grafana and developer dashboard methods take advantage of the Status API, including some endpoints that are also a tech preview.
• **Getting started with Graphite** on page 341

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. Once 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 345

These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

• **Using the developer dashboard** on page 349

While not as customizable as a Graphite dashboard, the developer dashboard is a simple, built-in way to view information about Puppet Server at a glance.

### 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. Once 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 Puppet master, because the module makes security policy changes that are inappropriate for a Puppet master:

• 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. As root on the Puppet agent node, run `puppet module install puppetlabs-grafanadash`.

3. As root on the Puppet agent node, run `puppet apply -e 'include grafanadash::dev'`. © 2020 Puppet, Inc.
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'S 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 master.example.com with the hostname of your Puppet master. 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 will remain empty until you enable Puppet Server's Graphite metrics.

Related information
Sample Grafana dashboard graphs on page 342
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 Classification, and in the PE Infrastructure group, select the PE Master group.
2. On the Configuration 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 Puppet master 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.

Note: In the Grafana UI, choose an appropriate time window from the drop-down menu.

Note: The puppet_enterprise::profile::master::metrics_enabled parameter used in Puppet Enterprise 2016.3 and earlier is no longer necessary and has been deprecated. If you set it, PE will notify you of the setting's deprecation.

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.
<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 once 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 compile masters to distribute the catalog compilation load.</td>
</tr>
<tr>
<td>JRuby Timers</td>
<td>This graph tracks several JRuby pool metrics.</td>
</tr>
<tr>
<td></td>
<td>• The borrow time represents the mean amount of time that Puppet Server uses (&quot;borrows&quot;) each JRuby from the pool.</td>
</tr>
<tr>
<td></td>
<td>• The wait time represents the total amount of time that Puppet Server waits for a free JRuby instance.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• The lock wait time represents the amount of time that Puppet Server waits to acquire a lock on the pool.</td>
</tr>
<tr>
<td></td>
<td>These metrics help identify sources of potential JRuby allocation bottlenecks.</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>This graph tracks how much heap and non-heap memory that Puppet Server uses.</td>
</tr>
<tr>
<td>Compilation</td>
<td>This graph breaks catalog compilation down into various phases to show how expensive each phase is on the master.</td>
</tr>
</tbody>
</table>

**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 `master.example.com`) and a way to add targets directly to an exported Grafana dashboard's JSON content.

```json
"panels": [ 
  {
    "span": 4,
```

© 2020 Puppet, Inc.
"editable": true,
"type": "graphite",
...

"targets": [
{
   "target": "alias(puppetlabs.master.example.com.num-cpus,'num cpus')"
},
{
   "target": "alias(puppetlabs.master.example.com.http.active-requests.count,'active requests')"
},
{
   "target": "alias(puppetlabs.master.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.<MASTER-HOSTNAME>; for instance, the Grafana dashboard file refers to the num-cpus metric as puppetlabs.<MASTER-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.<MASTER-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.<MASTER-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.<MASTER-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.<MASTER-HOSTNAME>.compiler.find_facts`: The time spent parsing facts.
- `puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.compiler.static_compile`: The time spent compiling static catalogs.
- `puppetlabs.<MASTER-HOSTNAME>.compiler.static_compile_inlining`: The time spent inlining metadata for static catalogs.
- `puppetlabs.<MASTER-HOSTNAME>.compiler.static_compile_postprocessing`: The time spent post-processing static catalogs.

File sync metrics:

- `puppetlabs.<MASTER-HOSTNAME>.file-sync-client.clone-timer`: The time spent by file sync clients on compile masters initially cloning repositories on the master of masters.
- `puppetlabs.<MASTER-HOSTNAME>.file-sync-client.fetch-timer`: The time spent by file sync clients on compile masters fetching repository updates from the master of masters.
- `puppetlabs.<MASTER-HOSTNAME>.file-sync-client.sync-clean-check-timer`: The time spent by file sync clients on compile masters checking whether the repositories are clean.
- `puppetlabs.<MASTER-HOSTNAME>.file-sync-client.sync-timer`: The time spent by file sync clients on compile masters synchronizing code from the private datadir to the live codedir.
- `puppetlabs.<MASTER-HOSTNAME>.file-sync-storage.commit-add-rm-timer`
- `puppetlabs.<MASTER-HOSTNAME>.file-sync-storage.commit-timer`: The time spent committing code on the master of masters into the file sync repository.

Function metrics:

- `puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.functions.<FUNCTION-NAME>.mean`.

HTTP metrics:

- `puppetlabs.<MASTER-HOSTNAME>.http.active-histo`: A histogram of active HTTP requests over time.
- `puppetlabs.<MASTER-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.<MASTER-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.<MASTER-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.<MASTER-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.<MASTER-HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-requests`
• `puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_content-/*/-requests`
• `puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_metadata-/*/-requests`
• `puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_metadatas-/*/-requests`

• `puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.http.puppet-v3-report-/*/-requests`: The time spent handling report requests. A bottleneck here might indicate an issue with PuppetDB.

• `puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.jruby.borrow-timer`: The time spent with a borrowed JRuby.

• `puppetlabs.<MASTER-HOSTNAME>.jruby.free-jrubies-histo`: A histogram of free JRubies over time. This metric's average value should be greater than 1; if it isn't, more JRubies or another compile master might be needed to keep up with requests.

• `puppetlabs.<MASTER-HOSTNAME>.jruby.lock-held-timer`: The time spent holding the JRuby lock.

• `puppetlabs.<MASTER-HOSTNAME>.jruby.lock-wait-timer`: The time spent waiting to acquire the JRuby lock.

• `puppetlabs.<MASTER-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.<MASTER-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.<MASTER-HOSTNAME>.puppetdb.catalog.save`
• `puppetlabs.<MASTER-HOSTNAME>.puppetdb.command.submit`
• `puppetlabs.<MASTER-HOSTNAME>.puppetdb.facts.find`
• `puppetlabs.<MASTER-HOSTNAME>.puppetdb.facts.search`
• `puppetlabs.<MASTER-HOSTNAME>.puppetdb.report.process`
• `puppetlabs.<MASTER-HOSTNAME>.puppetdb.resource.search`

Counters only

HTTP metrics:

• `puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.http.puppet-v3-catalog-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-environment-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-environment_classes-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-environments-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_content-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_metadata-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-file_metadatas-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-node-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-report-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-resource_type-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-resource_types-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-static_file_content-/*/-percentage
  • puppetlabs.<MASTER-HOSTNAME>.http.puppet-v3-status-/*/-percentage

  • puppetlabs.<MASTER-HOSTNAME>.http.total-requests: The total requests handled by Puppet Server.

  JRuby metrics:
  
  • puppetlabs.<MASTER-HOSTNAME>.jruby.borrow-count: The number of successfully borrowed JRubies.
  • puppetlabs.<MASTER-HOSTNAME>.jruby.borrow-retry-count: The number of attempts to borrow a JRuby that must be retried.
  • puppetlabs.<MASTER-HOSTNAME>.jruby.borrow-timeout-count: The number of attempts to borrow a JRuby that resulted in a timeout.
  • puppetlabs.<MASTER-HOSTNAME>.jruby.request-count: The number of requested JRubies.
  • puppetlabs.<MASTER-HOSTNAME>.jruby.return-count: The number of JRubies successfully returned to the pool.
  • puppetlabs.<MASTER-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 compile masters.
  • puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.num-cpus: The number of available CPUs on the server.
  • puppetlabs.<MASTER-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.<MASTER-HOSTNAME>.memory.total.committed
  puppetlabs.<MASTER-HOSTNAME>.memory.total.init
  puppetlabs.<MASTER-HOSTNAME>.memory.total.used
  puppetlabs.<MASTER-HOSTNAME>.memory.total.max
  puppetlabs.<MASTER-HOSTNAME>.memory.heap.committed
  puppetlabs.<MASTER-HOSTNAME>.memory.heap.init
  puppetlabs.<MASTER-HOSTNAME>.memory.heap.used
  puppetlabs.<MASTER-HOSTNAME>.memory.heap.max
  puppetlabs.<MASTER-HOSTNAME>.memory.non-heap.committed
  puppetlabs.<MASTER-HOSTNAME>.memory.non-heap.init
  puppetlabs.<MASTER-HOSTNAME>.memory.non-heap.used
  puppetlabs.<MASTER-HOSTNAME>.memory.non-heap.max

Related information

Modifying exported metrics on page 349

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.

Using the developer dashboard

While not as customizable as a Graphite dashboard, the developer dashboard is a simple, built-in way to view information about Puppet Server at a glance.

Important: The developer dashboard was removed in Puppet Server 5.3.7, and is not included in PE 2018.1.7 and newer versions. To gather Puppet Server metrics, use the puppet_metrics_collector module.

The developer dashboard features metrics particularly relevant to developers of Puppet manifests and modules, which are drawn from the Status API's metrics endpoints.

The dashboard charts the current and mean number of free and requested JRuby interpreters, as well as the mean JRuby borrow and wait times in milliseconds. It also lists the top 10 aggregate API endpoint requests, function calls, and resource declarations by total, mean, and aggregate counts. For more information about these metrics, see the documentation for the metrics endpoints.

© 2020 Puppet, Inc.
Accessing the developer dashboard
The developer dashboard, a tech preview, visualizes output from the Status API for reference when developing and deploying Puppet modules.

The developer dashboard is enabled by default in Puppet Enterprise. To access the developer dashboard:

Open a web browser and go to https://<DNS NAME OF YOUR MASTER>:8140/puppet/experimental/dashboard.html

Status API

The status API allows you to check the health of PE components. It can be useful in automated monitoring of your PE infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or when troubleshooting problems in PE.

You can check the overall health of pe-console-services, as well as the health of the individual services within pe-console-services:

- Activity service
- Node classifier
- PuppetDB
- Puppet Server
- Role-based access control (RBAC)
- Code Manager (if configured)

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 Puppet master.

- Authenticating to the status API on page 350
  Web session authentication is not required to access the status API. You can choose to authenticate requests by using whitelisted certificates, or can access the API without authentication via HTTP.

- Forming requests to the status API on page 351
  The HTTPS status endpoints are available on the console services API server, which uses port 4433 by default.

- JSON endpoints on page 351
  These two endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

- Activity service plaintext endpoints on page 353
  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.

- Metrics endpoints on page 354
  Puppet Server is capable of tracking advanced metrics to give you additional insight into its performance and health.

- The metrics API on page 360
  Puppet Enterprise includes an optional, enabled-by-default web endpoint for Java Management Extension (JMX) metrics, namely managed beans (MBeans).

Authenticating to the status API

Web session authentication is not required to access the status API. You can choose to authenticate requests by using whitelisted certificates, or can access the API without authentication via HTTP.

You can authenticate requests using a certificate listed in RBAC's certificate whitelist, located at /etc/puppetlabs/console-services/rbac-certificate-whitelist. 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.

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

The HTTPS status endpoints are available on the console services API server, which uses port 4433 by default.

The path prefix is /status, so, for example, the URL to get the statuses for all services as JSON is https://<DNS NAME OF YOUR CONSOLE HOST>:4433/status/v1/services.

To access that URL using curl commands, run:

```bash
curl https://<DNS NAME OF YOUR CONSOLE HOST>:4433/status/v1/services \
   --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem \
   --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem \
   --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem
```

If enabled, the HTTP status endpoints are available on port 8123.

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.

JSON endpoints

These two 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 all PE services.

The content type for this endpoint is application/json; charset=utf-8.

**Query parameters**

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>

**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>Can be critical, debug, or info.</td>
</tr>
<tr>
<td>state</td>
<td>Can be 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>
</tbody>
</table>
### active_alerts

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.

For example:

```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": [], "service_name": "rbac-service"
}

"classifier-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": [], "service_name": "classifier-service"
}
```

### 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 a particular PE service.

The content type for this endpoint is application/json; charset=utf-8.

### Query parameters

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>
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>Can be critical, debug, or info.</td>
</tr>
<tr>
<td>state</td>
<td>Can be 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:

```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

Activity service plaintext endpoints

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

Metrics endpoints
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 master server:

```
curl -k https://<DNS NAME OF YOUR MASTER>:8140/status/v1/services?level=debug
```

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><strong>Key</strong></td>
<td><strong>Definition</strong></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------</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 <code>jruby-pool-lock-status/current-state</code> 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 <code>num-pool-locks</code>.</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>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>• <code>duration-millis</code>: The length of time that the instance has been running.</td>
</tr>
<tr>
<td></td>
<td>• <code>reason/request</code>: A hash of details about the request being served.</td>
</tr>
<tr>
<td></td>
<td>• <code>request-method</code>: The HTTP request method, such as POST, GET, PUT, or DELETE.</td>
</tr>
<tr>
<td></td>
<td>• <code>route-id</code>: The route being served. For routing metrics, see the HTTP metrics endpoint.</td>
</tr>
<tr>
<td></td>
<td>• <code>uri</code>: The request’s full URI.</td>
</tr>
<tr>
<td></td>
<td>• <code>time</code>: 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>requested-instances</td>
<td>A list of the requests waiting to be served, each reporting:</td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration-millis</td>
<td>The length of time the request has waited.</td>
</tr>
<tr>
<td>reason/request</td>
<td>A hash of details about the waiting request.</td>
</tr>
<tr>
<td>request-method</td>
<td>The HTTP request method, such as POST, GET, PUT, or DELETE.</td>
</tr>
<tr>
<td>route-id</td>
<td>The route being served. For routing metrics, see the HTTP metrics endpoint.</td>
</tr>
<tr>
<td>uri</td>
<td>The request’s full URI.</td>
</tr>
<tr>
<td>time</td>
<td>The time (in milliseconds since the Unix epoch) when Puppet Server received the request.</td>
</tr>
</tbody>
</table>

#### return-count

The total number of JRuby instances that have been used.

For example:

```json
"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/
              }
            },
            "time": 1448478371406
          }
        ],
        "num-free-jrubies": 0,
        "num-jrubies": 1,
        "num-pool-locks": 2849,
        "requested-count": 640,
        "requested-instances": [
          {
```
"duration-millis": 3663,
"reason": {
   "request": {
      "request-method": "put",
      "route-id": "puppet-v3-report-/*/",
      "uri": "/puppet/v3/report/
      hostname.example.com"
   }
   "time": 1448478371715
   }
   },
   "return-count": 638
   }
   
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:

"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]"
        },
        {...},
      ]
    }
  }
}
```

### The 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:

```
curl 'http://localhost:8080/metrics/v1/mbeans/java.lang:type=Memory'
```

The response should contain a JSON object representing the data:

```
{
    "ObjectPendingFinalizationCount" : 0,
    "HeapMemoryUsage" : {
        "committed" : 807403520,
        "init" : 268435456,
        "max" : 3817865216,
        "used" : 129257096
    },
    "NonHeapMemoryUsage" : {
        "committed" : 85590016,
        "init" : 24576000,
        "max" : 184549376
    }
}
```
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 nodes** on page 362
  To manage nodes with Puppet Enterprise (PE), you must approve the node’s certificate signing request. If you no longer wish to manage a node, you can remove all traces of it from PE

- **Running Puppet on nodes** on page 364
  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 365
  Configure nodes by assigning classes, parameters, and variables to them. This is called classification.

- **Making changes to node groups** on page 373
  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 374
  An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

- **Preconfigured node groups** on page 375
  Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

- **Designing system configs: roles and profiles** on page 379
  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 service API** on page 404
  These are the endpoints for the node classifier v1 API.

## Adding and removing nodes

To manage nodes with Puppet Enterprise (PE), you must approve the node’s certificate signing request. If you no longer wish to manage a node, you can remove all traces of it from PE

## Managing certificate signing requests

When you install a new PE agent, the agent automatically submits a certificate signing request (CSR) to the master.

Certificate requests can be signed from the console or the command line. If DNS altnames are set up for agent nodes, you must use the command line interface to approve and reject node requests.

After approving a node request, the node doesn’t show up in the console until the next Puppet run, which can take up to 30 minutes. You can manually trigger a Puppet run if you want the node to appear immediately.

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**.
Managing certificate signing requests in the console
The console displays a list of nodes on the Unsigned certs page that have submitted CSRs. You can approve or deny CSRs individually or in a batch.
If you use the Accept All or Reject All options, processing could take up to two seconds per request.
When using 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.

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 puppet cert list
```

To sign a pending request:

```
$ sudo puppet cert sign <name>
```

To sign pending requests for nodes with DNS altnames:

```
$ sudo puppet cert sign (<HOSTNAME> or --all) --allow-dns-alt-names
```

Remove nodes
To completely remove a node from PE, you must purge the node and revoke its certificate so that it doesn’t continue to check in.
Removing a node:

- Deactivates the node in PuppetDB.
- Deletes the Puppet master’s information cache for the node.
- Frees up the license that the node was using.
- Allows you to re-use the hostname for a new node.

**Note:** Purging a node doesn’t uninstall the agent from the node.

1. On the agent node, stop the agent service.
   - Agent versions 4.0 or later: `service puppet stop`
   - Agent versions earlier than 4.0: `service pe-puppet stop`
   **Note:** You can run `puppet --version` to see which version of Puppet you’re using.
2. On the master, purge the node: `puppet node purge <CERTNAME>`
   The node’s certificate is revoked, the certificate revocation list (CRL) is updated, and the node is deactivated in PuppetDB and removed from the console, increasing your license count. The node can’t check in or re-register with PuppetDB on the next run.
3. If you have compile masters, run Puppet on them: `puppet agent -t`
   The updated CRL is managed by Puppet and distributed to compile masters.
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.
5. (Optional) If the node still exists but you no longer want to manage it, stop MCollective on the node:
   a) Uninstall the agent or stop the MCollective service:
      - Agent versions 4.0 or later: `service mcollective stop`
      - Agent versions earlier than 4.0: `service pe-mcollective stop`
   b) Remove the node’s certificate in `/etc/puppetlabs/mcollective/ssl/clients`.  

© 2020 Puppet, Inc.
Related information
Uninstall agents on page 221
You can remove the agent from nodes that you no longer want to manage.

POST /v1/commands/unpin-from-all on page 436
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.

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 master and agent nodes perform the following actions:
1. The agent node sends facts to the master and requests a catalog.
2. The master 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 in Puppet Enterprise on page 261
A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources on a node that have source attributes. Using static catalogs can reduce the number of requests an agent makes to the master.

Running Puppet with the orchestrator
The Puppet orchestrator is a set of interactive tools used to deploy configuration tools 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" })}'
• application or application instance - from the command line.

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 in the console on page 489
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 any node.

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:** If an agent does not have an active websocket session with the PCP broker, the Run Puppet button is disabled.

**Related information**

- [Node run statuses](#) on page 327
- The Overview 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 321
- 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>
<tr>
<td>Report request times out</td>
<td>Occurs when the report is not successfully stored in PuppetDB after the run completes.</td>
</tr>
<tr>
<td>Invalid response (such as a 500 error)</td>
<td>Your Puppet code might be have incorrect formatting.</td>
</tr>
<tr>
<td>Button is disabled and a run is not allowed.</td>
<td>The user has permission, but the agent is not responding.</td>
</tr>
</tbody>
</table>

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 Classification 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 440
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.

Create node groups
You can create node groups to assign either an environment or classification.

- Environment node groups assign environments to nodes, such as test, development, or production.
- Classification node groups assign classification data to nodes, including classes, parameters, and variables.

Create environment node groups
Create custom environment node groups so that you can target deployment of Puppet code.

Note: The "All environments" node group and its child groups "Production environment", "Development environment", and "Development one-time run exception" appear only in PE 2018.1.7 and newer.

1. In the console, click Classification, 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.

### Create classification node groups

Create classification node groups to assign classification data to nodes.

1. In the console, click **Classification**, 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 **Classification**, 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 Classification, 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 may 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 os.release.major or trusted.certname.</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 processors.models[0] or mountpoints./.options[0].</td>
</tr>
<tr>
<td></td>
<td>• To identify path segments that contain dots or UTF-8 characters, surround the segment with single or double quotes, such as trusted.extensions.&quot;1.3.6.1.4.1.34380.1.2.1&quot;.</td>
</tr>
<tr>
<td></td>
<td>• To use trusted extension short names, append the short name after a second dot, such as trusted.extensions.pp_role.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> 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.</td>
</tr>
<tr>
<td>Option</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Operator</strong></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>The numeric operators &gt;, &gt;=, &lt;, and &lt;= can be used only with facts that have a numeric value.</td>
<td></td>
</tr>
<tr>
<td>To match highly specific node facts, use ~ or !~ with a regular expression for <strong>Value</strong>.</td>
<td></td>
</tr>
</tbody>
</table>

| Value | Specifies the value associated with the fact. |

### 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 **Classification**, and then find the node group that you want to add the class to and select it.
2. On the **Configuration** tab, in the **Add new class** field, select the class to add.

   The **Add new class** field suggests classes that the master 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 master 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 master immediately, click the **Refresh** button.

© 2020 Puppet, Inc.
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.

On your master, 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, you might need to add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

If your environment is configured for high availability, you must also update `hiera.yaml` on your replica.

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 the console 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.

1. In the console, click **Classification**, then find the node group that you want to add configuration data to and select it.
2. On the **Configuration** tab in the **Data** section, 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 244

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 **Classification**, and then find the node group that you want to add a parameter to and select it.

© 2020 Puppet, Inc.
2. On the **Configuration** tab, in the **Classes** section, 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 **Classification**, 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.

- **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.

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]}`  
- `${osrelease}`  

In the console, an index can only be 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 **Classification**, 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.

© 2020 Puppet, Inc.
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 Classification, 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 Classification, 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 436

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 Classification, and select a node group.
2. On the Configuration tab in the Classes section, 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 Classification, and select a node group.
2. On the **Configuration** tab in the **Classes** section, for the class and parameter that you want to edit, select an option.
   - **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 **Classification**, 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.

Before testing and promoting data using an environment-based workflow, you must have configured:
- A test environment that's a child of the production environment
- Classification node groups that include nodes assigned dynamically or statically

**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. (Optional) 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. If you're satisfied with your test results, change the environment for the node group from test to production.

**Testing code with canary nodes using alternate environments**
Puppet Enterprise allows you to centrally manage which nodes are in which environments.

In most cases the environments are long-lived, such as development, testing, and production, and nodes don't move between these environments after their initial environment has been set.

When an agent node matches the rules specified in environment node groups, the agent is classified in that environment regardless of any environments specified in the agent's own `puppet.conf` file. Agents can't override this server-specified environment. That's the desired behavior in most cases.
A notable exception is when you want to test new Puppet code before deployment, and you have a code promotion workflow based on environments. In this case, you can specify that certain nodes are allowed to use an agent-specified environment. You map the agent-specified environment to a feature branch in your version control system. This override enables you to quickly test the code in your feature branch without permanently changing the environment that the node is in.

To apply an agent-specified environment for more than one run, specify the environment in the node's puppet.conf file. Doing this also sets the agent_specified_environment fact to true. The node will continue to get the agent-specified environment until you remove the environment from its puppet.conf file, or change the rules in the testing environment group.

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 compile masters. 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 only used 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.

In a monolithic install, <YOUR HOST> is your master certname. You can find the certname with puppet config print certname. In a split install, <YOUR HOST> is the certname of the server on which you installed the component.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcollective_middleware_hosts</td>
<td>&quot;[&quot;&lt;YOUR HOST&gt;&quot;]&quot;</td>
</tr>
<tr>
<td></td>
<td>This value must be an array, even if</td>
</tr>
<tr>
<td></td>
<td>there is only one value, for example</td>
</tr>
<tr>
<td></td>
<td>mcollective_middleware_hosts =</td>
</tr>
<tr>
<td></td>
<td>[&quot;master.testing.net&quot;]</td>
</tr>
<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</td>
</tr>
<tr>
<td></td>
<td>number from the default 5432.</td>
</tr>
<tr>
<td>database_ssl</td>
<td>true if you're using the PE-installed</td>
</tr>
<tr>
<td></td>
<td>PostgreSQL, and false if you're using your</td>
</tr>
<tr>
<td></td>
<td>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>console_port</td>
<td>&quot;&lt;YOUR PORT NUMBER&gt;&quot;</td>
</tr>
<tr>
<td></td>
<td>Required only if you changed the port</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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 first master node

**Matching nodes**

On a new install, the master is pinned to this node group.

**Notes**

Don't add additional nodes to this node group.

**PE MCollective node group**

This node group is used to enable the MCollective engine on all matching nodes.

**Classes**

puppet_enterprise::profile::mcollective::agent — manages the MCollective server
Matching nodes
   All nodes.

Notes
   You might have some nodes, such as non-root nodes or network devices, that should not have MCollective enabled. You can create a rule in this node group to exclude these nodes.

**PE Master node group**
This node group is used to manage masters and add compile masters.

Classes
   - puppet_enterprise::profile::master — manages the Puppet master service
   - puppet_enterprise::profile::mcollective::peadmin — manages the peadmin MCollective client
   - puppet_enterprise::profile::master::mcollective — manages keys used by MCollective

Matching nodes
   On a new install, the master is pinned to this node group.

Related information
**Install compile masters** on page 207
To install a compile master, you first install an agent and then classify that agent as a compile master.

**PE Orchestrator node group**
This node group is used to manage the application orchestration service.

Classes
   - puppet_enterprise::profile::orchestrator — manages the application orchestration service

Matching nodes
   On a new install, the master 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 the database service.

Classes
   - puppet_enterprise::profile::puppetdb — manages the PuppetDB service

Matching nodes
   On a new install, the PuppetDB server node is 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 ActiveMQ Broker node group
This node group is used to manage the ActiveMQ broker and add additional ActiveMQ brokers.

Classes

- puppet_enterprise::profile::amq::broker — manages the ActiveMQ Collective broker

Matching nodes
On a new install, the master is pinned to this node group.

Related information

Install ActiveMQ hubs and spokes on page 209
Setting up hubs and spokes involves classifying nodes into appropriate node groups, and then configuring the connections for those node groups.

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 master, PuppetDB, console, and compile masters.

Notes
You might want to manually pin to this group any additional nodes used to run your infrastructure, such as compile master load balancer nodes. Pinning a compile master load balancer node to this group allows it to receive its catalog from the master of masters, rather than the compile master, 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.

Environment node groups
Environment node groups are used only to set environments. They should not contain any classification.

Preconfigured environment node groups differ depending on your version of PE, and you can customize environment groups as needed for your ecosystem.
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 379
  The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.
- Roles and profiles example on page 382
  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 384
  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 401
  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:

```ruby
class role::jenkins::master {
    include profile::base
    include profile::server
    include profile::jenkins::master
}
```

- Each profile configures a layered technology stack, using multiple component modules and the built-in resource types. (In the diagram, profile::jenkins::master uses rtyler/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 will always use 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.

```yaml
# Example Hiera data
profile::jenkins::jenkins_port: 8000
profile::jenkins::java_dist: jre
profile::jenkins::java_version: '8'

# Example manifest
class profile::jenkins (  
  Integer $jenkins_port,  
  String $java_dist,  
  String $java_version  
) {  
  # ...
}
```

This profile omits the parameters and uses the `lookup` function:

```yaml
class profile::jenkins {  
  $jenkins_port = lookup('profile::jenkins::jenkins_port', {value_type => String, default_value => '9091'})  
  $java_dist = lookup('profile::jenkins::java_dist', {value_type => String, default_value => 'jdk'})  
  $java_version = lookup('profile::jenkins::java_version', {value_type => String, default_value => 'latest'})  
  # ...
}
```

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. The standard module for that is rtyler/jenkins.

Jenkins requires Java, and the rtyler 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 rtyler/jenkins, puppetlabs/java.

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,  
}
```

© 2020 Puppet, Inc.
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 380

There are rules for writing profile classes.

*Methods for data lookup* on page 381

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.

**Tip:** In most cases, setting configuration data in 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 data. Data set in the console overrides 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 .
- You simply prefer the console user interface.

```
# /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.

```puppet
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

Related information
Rules for role classes on page 381
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.

```puppet
lookup('classes', {merge => unique}).include
```

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'll explain our choices and point out some of the common trade-offs you'll 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 },
  },
}
Related information

Rules for profile classes on page 380
There are rules for writing profile classes.

First refactor: 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 once, 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 should be on Jenkins machines. After looking at our past usage, we realized that we only use 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:

class profile::jenkins::master (  
    String $jenkins_port = '9091',  
    Boolean $install_jenkins_java = true,  
) { # ...

And here's how we choose which Java to use:

class { 'jenkins':  
    configure_firewall => true,  
    install_java => $install_jenkins_java,  
    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:

::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}": }
    }
  }
  }
  default: {  
    notify { "profile::jenkins::usage::java8 cannot set up JDK on
        ${::osfamily}": }
  }

  }  
  }
}

Diff of first refator

@@ -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',  
  String $java_version = 'latest',  
+  String  $jenkins_port = '9091',  
+  Boolean $install_jenkins_java = true,  
) {
  class { 'jenkins':
    configure_firewall => true,
    - install_java       => false,
+ install_java => $install_jenkins_java,
    port               => $jenkins_port,
    config_hash        => {
      'HTTP_PORT' => { 'value' => $jenkins_port },
    },
+  # When not using the jenkins module's java version, install java8.
+  unless $install_jenkins_java  { include profile::jenkins::usage::java8 }
}  

@@ -15,9 +14,6 @@ class profile::jenkins::master (  
  }
  }
  }

- class { 'java':
-   distribution => $java_dist,
-   version => $java_version,
-   before => Class['jenkins'],
-  }
+ # When not using the jenkins module's java version, install java8.
+ unless $install_jenkins_java  { include profile::jenkins::usage::java8 }

© 2020 Puppet, Inc.
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 we'll use it:

```bash
# 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', $::memorysize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
# system+properties
jenkins::sysconfig { 'JAVA_ARGS':
    value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
}
```

Note: Rule 4 again — we couldn't hardcode this, because we have some smaller Jenkins masters that can't spare the extra memory. But since 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

```bash
@@ -16,4 +16,20 @@ class profile::jenkins::master (
    # When not using the jenkins module's java version, install java8.
    unless $install_jenkins_java { include profile::jenkins::usage::java8 }
+    # Manage the heap size on the master, in MB.
+    if($::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', $::memorysize_mb * 0.5)
+    }
+    # Set java params, like heap min and max sizes. See
+    # https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+    # system+properties
+    jenkins::sysconfig { 'JAVA_ARGS':
+        value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
+    }
+}
```

© 2020 Puppet, Inc.
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 will get the new Jenkins version first, and we can ensure everything works as expected before upgrading our prod machines.

class 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,
  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/manifests/jenkins/master.pp
class profile::jenkins::master {
-  String  $jenkins_port = '9091',
-  Boolean $install_jenkins_java = true,
+  String                      $jenkins_port = '9091',
+  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
+  Boolean                     $install_jenkins_java = true,
  }{
    class { 'jenkins':
      lts => true,
      repo => true,
      direct_download => $direct_download,
      version => 'latest',
      service_enable => true,
      service_ensure => running,
      configure_firewall => true,
      install_java => $install_jenkins_java,
      port => $jenkins_port,
    }
  }

© 2020 Puppet, Inc.
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: since 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'll change the Jenkins profile to work the same way; we'll manage the `jenkins` user alongside the rest of our user accounts. While we're doing that, we'll 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 you should 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.

```puppet
# 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
# 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 => true,
  install_java  => $install_jenkins_java,
  manage_user   => false,
}
```

© 2020 Puppet, Inc.
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**

```perls
@@ -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 
+  # '${master_config_dir}/plugins' here to prevent the upstream 
+  # rtyler-jenkins module from trying to manage the homedir as the config 
+  # dir. For more info, see the upstream module's 'manifests/plugin.pp' 
+  # manifest. 
+  file { "${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, 
```

© 2020 Puppet, Inc.
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.

Since 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.

```bash
$selfure_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",
}

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",
}
```

Plugins are also a bit tricky, because we have a few Jenkins masters where we want to manually configure plugins. So we'll put the base list in a separate profile, and use a parameter to control whether we use it.

```ruby
class profile::jenkins::master {
   Boolean $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
# /etc/puppetlabs/code/environments/production/site/profile/manifests/jenkins/master/plugins.pp
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 (String                      $jenkins_port = '9091',
                              +  Boolean                     $manage_plugins = false,
                              Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
                              Boolean                     $install_jenkins_java = true,
                              ) {
@@ -14,6 +15,20 @@ class profile::jenkins::master (String                      $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
  +  # 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 (String                      $jenkins_group          = $profile::jenkins::params::jenkins_group
                                                  $master_config_dir      = $profile::jenkins::params::master_config_dir
  +  # Deploy the SSH keys that Jenkins needs to manage its agent machines and
  +  # access Git repos.
  +  file { "${master_config_dir}/.ssh":
  +    ensure => directory,
```
Sixth refactor: Manage logging and backups

Backing 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.

```puppet
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’ll 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 will log there instead of its own file.

We’ll use jenkins::sysconfig and our homegrown logrotate::job to do the work:

```puppet
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 7',
'compress',
'delaycompress',
'notifempty']
}

Diff of sixth refactor

@@ -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 => 
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
Dhudson.model.DirectoryBrowserSupport.CSP=\\"default-src "self"; img-src "self"; style-src "self";\\\",
}
+
+    # Forward jenkins master logs to syslog.
+    + # When set to facility.level the jenkins_log will use that value instead
+    + # 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 { "$({master_config_dir})/ssh":
@@ -109,4 +120,29 @@ class profile::jenkins::master (source => "puppet://$(secure_server)/secure/delivery/id_rsa-jenkins.pub",
    }
+
+    +    # Back up Jenkins' data.
+    + if $backups_enabled {
+      +      backup::job { "jenkins-data-$(::hostname)"
+      +      files => $master_config_dir,
+      +    }
+    + }

© 2020 Puppet, Inc.
Logrotate rules:
Jenkins' default logrotate config retains too much data: by default, it rotates jenkins.log weekly and retains the last 52 weeks of logs. Considering we almost never look at the logs, let's rotate them daily and discard after 7 days to reduce disk usage.

```
logrotate::job { 'jenkins':
  log     => '/var/log/jenkins/jenkins.log',
  options => [ 'daily',
               'copytruncate',
               'missingok',
               'rotate 7',
               'compress',
               'delaycompress',
               'notifempty',
  ],
}
```

Seventh refactor: Use a reverse proxy for HTTPS
We want the Jenkins web interface to use HTTPS, which we'll accomplish with an Nginx reverse proxy. We'll also standardize the ports: the Jenkins app will always bind to its default port, and the proxy will always serve over 443 for HTTPS and 80 for HTTP.

If we want to keep vanilla HTTP available, we'll provide an $ssl parameter. If set to false (the default), you can access Jenkins via both HTTP and HTTPS. We'll also add a $site_alias parameter, so the proxy can listen on a hostname other than the node's main FQDN.

```
class profile::jenkins::master (  
  Boolean                     $ssl = false,
  Optional[String[1]]         $site_alias = undef,
  # IMPORTANT: notice that $jenkins_port is removed.
  # ...
)
```

We'll set configure_firewall => false in the Jenkins class:

```
class { 'jenkins':
  lts                => true,
  repo               => true,
  direct_download    => $direct_download,
  version            => 'latest',
  service_enable     => true,
  service_ensure     => running,
  configure_firewall => false,    # <-- here
  install_java       => $install_jenkins_java,
  manage_user        => false,
  manage_group       => false,
  manage_datadirs    => false,
  # IMPORTANT: notice that port and config_hash are removed.
}
```

We need to deploy SSL certificates where Nginx can reach them. Since we serve a lot of things over HTTPS, we already had a profile for that:

```
# Deploy the SSL certificate/chain/key for sites on this domain.
include profile::ssl::deliveryWildcard
```
This is also a good time to add some info for the message of the day, handled by puppetlabs/motd:

```puppet
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}
  |
  order => 25,
}
}
```

The bulk of the work will be 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:

```puppet
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 will be reused by another profile, obey rule 1.

**Diff of seventh refactor**

```diff
@@ -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 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 {
    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
+ # 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 since 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 java8.

## 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
    # 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")
        }
    }
}
```

© 2020 Puppet, Inc.
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 since it's long, this class reads more cleanly with it separated out.

```puppet
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.

```puppet
$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).

```puppet
if $::osfamily == 'Debian' { include apt }
```

```puppet
package { 'git':
  ensure => present,
}
```

```puppet
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.

```puppet
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.

```puppet
unless $install_jenkins_java { include profile::jenkins::usage::java8 }
```

Manage the heap size on the master, in MB.

```puppet
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', $::memorysize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
    value => "-Xms$heapm -Xmx$heapm -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled
Dhudson.model.DirectoryBrowserSupport.CSP="\"default-src 'self'; img-src
'self'; style-src 'self';\"",
}
# Forward jenkins master logs to syslog.
# When set to facility.level the jenkins_log will use 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":
    ensure => file,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0640',
    source => "puppet://$secure_server/secure/delivery/id_rsa-jenkins.pub",
}
# Back up Jenkins' data.
if $backups_enabled {
    backup::job { "jenkins-data-$::hostname":
        files => $master_config_dir,
    }
}

# (QENG-1829) Logrotate rules:
# Jenkins' default logrotate config retains too much data: by default, it
# rotates jenkins.log weekly and retains the last 52 weeks of logs.
# Considering we almost never look at the logs, let's rotate them daily
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:

```puppet
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

Related information

Rules for role classes on page 381

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:

```puppet
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

...and an RE-specific one:

```puppet
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'll 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.

```
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.

```
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 only go one level deep, but it can get wild 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**

Since 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'll need 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 service API

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 405
Requests to the node classifier API must be well-formed HTTP(S) requests.
• Groups endpoint on page 406
The groups endpoint is used to create, read, update, and delete groups.
• Groups endpoint examples on page 422
Use example requests to better understand how to work with groups in the node classifier API.
• Classes endpoint on page 424
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 426
Use the classification endpoint takes a node name and a set of facts and returns information about how that
node should be classified. The output can help you test your classification rules.
• Commands endpoint on page 436
Use the commands endpoint to unpin specified nodes from all groups they’re pinned to.
• Environments endpoint on page 437
Use the environments endpoint to retrieve information about environments in the node classifier. The output
will either tell you 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 this endpoint should not be used to create, update, or delete them.
• **Nodes check-in history endpoint** on page 438
  Use the nodes endpoint to retrieve historical information about nodes that have checked into the node classifier.

• **Group children endpoint** on page 440
  Use the group children endpoint to retrieve a specified group and its descendents.

• **Rules endpoint** on page 444
  Use the rules endpoint to translate a group’s rule condition into PuppetDB query syntax.

• **Import hierarchy endpoint** on page 445
  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 446
  Use the last class update endpoint to retrieve the time that classes were last updated from the Puppet master.

• **Update classes endpoint** on page 446
  Use update classes endpoint to trigger the node classifier to update class and environment definitions from the Puppet master.

• **Validation endpoints** on page 447
  Use validation endpoints to validate groups in the node classifier.

• **Node classifier errors** on page 450
  Familiarize yourself with error responses to make working the node classifier service API easier.

### 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 and tuning the console](#) on page 242
  After installing Puppet Enterprise, you can change product settings to customize the console's behavior, adjust to your team's needs, and improve performance.

**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 RBAC certificate whitelist.

**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 this 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 and saved it as an environment variable using `export TOKEN=<PASTE THE TOKEN HERE>`.

```bash
curl -k -X GET https://<HOSTNAME>:<PORT>/classifier-api/v1/groups -H "X-Authentication:$TOKEN"
```

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
curl -k -X GET "https://<HOSTNAME>:<PORT>/classifier-api/v1/groups?token=$TOKEN"
```

**Note:** Supplying the token as a token parameter is not as secure as using the X-Authentication method.
Whitelisted certificate

You can also authenticate requests using a certificate listed in RBAC’s certificate whitelist. The RBAC whitelist is located at /etc/puppetlabs/console-services/rbac-certificate-whitelist. 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 whitelist certificate name and the private key to run the script.

The following query will return a list of all groups that exist in the node classifier, along with their associated metadata. This query shows how to attach the whitelist certificate to authenticate the node classifier API.

In this query, the “whitelisted certname” needs to match a name in the file, /etc/puppetlabs/console-services/rbac-certificate-whitelist.

```
curl -X GET https://<HOSTNAME>:<PORT>/classifier-api/v1/groups 
    --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem 
    --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem 
    --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem -H "Content-Type: application/json"
```

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 289

Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. 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 will 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 will get a 400 Bad Request error.

The following example shows a request using the limit and offset parameters.

```
curl https://<DNS NAME OF CONSOLE>:4433/classifier-api/v1/groups?
    limit=10&offset=20 
    --cert /etc/puppetlabs/puppet/ssl/certs/<WHITELISTED CERTNAME>.pem 
    --key /etc/puppetlabs/puppet/ssl/private_keys/<WHITELISTED CERTNAME>.pem 
    --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem -H "Content-Type: application/json"
```

The 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 ([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 will 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 will not be 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 web servers, 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, since its deleted parameters object's `puppetlabs.classifier/deleted` key maps to `true`, in contrast to the `rocket` class, which has only had its `stages` parameter deleted.

### Rule condition grammar

Nodes can be classified into groups using rules. This example shows how rule conditions must be structured:

```
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**

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:
**Key** | **Definition**
---|---
kind | "schema-violation"

details | An object that contains three keys:
  • submitted — Describes the submitted object.
  • schema — Describes the schema that object should 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 these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
</table>
| kind | "malformed-request"
| 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 these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
</table>
| 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 these keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
</table>
| kind | "missing-referents"
| msg | Describes the error and lists the missing classes or parameters.
An array of objects, where each object describes a single missing referent, and has the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>&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>missing</td>
<td>The name of the missing class or class parameter.</td>
</tr>
<tr>
<td>environment</td>
<td>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>group</td>
<td>The name of the node group where the error was encountered. Note that, due to inheritance, this may not be the group where the parameter was defined.</td>
</tr>
<tr>
<td>defined_by</td>
<td>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>
<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>

### 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>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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 will 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>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>
</tbody>
</table>
### Key Definitions

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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 will not be 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 will be 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 450
- 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.

**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>
<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>
</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>
<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>

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 will be merged with the node group, and then any keys of the resulting object that have a null value will be 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 will be updated wholesale or removed from the group accordingly.

The name, environment, description, and parent keys, if present in the delta, will 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 will 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:

{
  "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:

{
  "name": "Production Webservers",  
  "id": "58463036-0efa-4365-b367-b5401c0711d3",  
  "environment": "production",  
  "parent": "01522c99-627c-4a07-b28e-a25dd563d756",  
  "rule": ["~", ["trusted", "certname"], "www"],  
  "classes": {  
    "apache": {  
      "serveradmin": "roy@reynholm.co.uk"  
    },  
    "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</td>
</tr>
<tr>
<td></td>
<td>request body.</td>
</tr>
<tr>
<td></td>
<td>• fromUrl — Contains the ID taken from the request</td>
</tr>
<tr>
<td></td>
<td>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 server 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 450
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:
  
  ```json
  {"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 <code>nodes</code> 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 <code>body</code> key containing the body as received by the service, and an <code>error</code> 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:
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:
  ```
  {"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:

<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 <code>nodes</code> query parameter.</td>
</tr>
</tbody>
</table>

If a request body is supplied but it is not valid JSON, the service will return 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>
</tbody>
</table>

© 2020 Puppet, Inc.
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>

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 master is running on puppetlabs-nc.example.vm with access to certificates and whitelisting 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.

```
curl -X POST -H 'Content-Type: application/json' \
 --cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem \
 --key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem \
 --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
 -d '{ "name": "My Nodes", "parent": "00000000-0000-4000-8000-000000000000", "environment": "production", "classes": {} }' \
 https://puppetlabs-nc.example.vm:4433/classifier-api/v1/groups
```

Get the group ID of My Nodes

This request uses the groups endpoint to get details about groups.

```
curl 'https://puppetlabs-nc.example.vm:4433/classifier-api/v1/groups' \
 -H 'Content-Type: application/json' \
 --cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem \
 --key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem \
 --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem
```

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 406
The groups endpoint is used to create, read, update, and delete groups.

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.

```
curl -X POST -H 'Content-Type: application/json' \
  --cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem \
  --key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem \
  --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
  -d '{ "nodes": ["example-to-pin.example.vm"] }' \
  https://puppetlabs-nc.example.vm:4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin
```

Related information

POST /v1/groups/<id> on page 417
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.

Note: You must supply all the nodes to pin to the group. For example, this request includes both example-to-pin.example.vm and example-to-pin-2.example.vm.

```
curl -X POST -H 'Content-Type: application/json' \
  --cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem \
  --key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem \
  --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
  -d '{ "nodes": ["example-to-pin-2.example.vm"] }' \
  https://puppetlabs-nc.example.vm:4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin
```

Related information

POST /v1/groups/<id> on page 417
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.

```
curl -X POST -H 'Content-Type: application/json' \
  --cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem \
  --key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem \
  --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
  -d '{ "nodes": ["example-to-unpin.example.vm"] }' \
  https://puppetlabs-nc.example.vm:4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin
```
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.

curl -X POST -H 'Content-Type: application/json'  
--cert /etc/puppetlabs/puppet/ssl/certs/puppetlabs-nc.example.vm.pem.pem  
--key /etc/puppetlabs/puppet/ssl/private_keys/puppetlabs-nc.example.vm.pem.pem  
--cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem  
-d '{ "classes": { "apache": { "serveradmin": "roy@reynholm.co.uk", "keepalive_timeout": null} } }'  
https://puppetlabs-nc.example.vm.pem:4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65

Related information
POST /v1/groups/<id> on page 417
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 master. To force an update, use the update_classes endpoint. To determine when classes were last retrieved from the master, 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. 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, or null. If the value is null, 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. 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, 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>
</tbody>
</table>
### Key
<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parameters</strong></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, 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

Use the classification endpoint takes a node name and a set of facts and returns information about how that node should be 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fact</strong></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><strong>trusted</strong></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 <code>fact</code> 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><strong>name</strong></td>
<td>The name of the node (a string).</td>
</tr>
<tr>
<td><strong>groups</strong></td>
<td>An array of the node groups (UUID strings) that this node was classified into.</td>
</tr>
<tr>
<td><strong>environment</strong></td>
<td>The name of the environment that this node will use, which is taken from the node groups the node was classified into.</td>
</tr>
<tr>
<td><strong>classes</strong></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><strong>parameters</strong></td>
<td>An object where the keys are top-level variable names and the values are the values that will be assigned to those variables.</td>
</tr>
</tbody>
</table>
This is an example of a response from this endpoint:

```json
{
    "name": "foo.example.com",
    "groups": ["00000000-0000-4000-8000-000000000000", "08c7915b-b83f-4d11-9522-6a60e2378cef"],
    "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:

<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 will be 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 must not define the value, because it might 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",
    "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",
                            "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" node group (since 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 450
  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:
### Key Definition

<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</td>
</tr>
<tr>
<td></td>
<td>a further object whose keys are fact names, and whose values are the</td>
</tr>
<tr>
<td></td>
<td>corresponding fact values. Structured facts can be included here;</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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:
### Key definitions

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>match_explanations</code></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><code>leaf_groups</code></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><code>inherited_classifications</code></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><code>conflicts</code></td>
<td>Corresponds to step 4 of classification. This key is present only if there are conflicts between the inherited classifications. Its value will be similar to a classification object, but wherever there was a conflict there will be 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><code>individual_classification</code></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><code>final_classification</code></td>
<td>Corresponds to step 6 of classification, will only be present if there are no conflicts between the inherited classifications. Its value will be the result of merging all individual node classification and group classification.</td>
</tr>
<tr>
<td><code>node_as_received</code></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><code>classification_source</code></td>
<td>An annotated version of the classification that has the environment and every class parameter and variable replaced with an &quot;annotated value&quot; 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",
```
"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"]
      },
      {
        "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": {
    "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"}
    }
  }
}
"config_data": {
    "USS::Voyager": {"designation": "subsequent"}
}
"inherited_classifications": {
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
        "environment": "alpha-quadrant",
        "variables": {},
        "classes": {
            "logic": {"importance": "primary"},
            "emotion": {"importance": "ignored"}
        },
        "config_data": {
            "USS::Enterprise": {"designation": "original"},
            "USS::Voyager": {"designation": "subsequent"}
        }
    }
},
"individual_classification": {
    "classes": {
        "emotion": {
            "importance": "secondary"
        }
    },
    "variables": {
        "full_name": "S'chn T'gai Spock"
    }
},
"final_classification": {
    "environment": "alpha-quadrant",
    "variables": {
        "full_name": "S'chn T'gai Spock"
    },
    "classes": {
        "logic": {"importance": "primary"},
        "emotion": {"importance": "secondary"}
    },
    "config_data": {
        "USS::Enterprise": {"designation": "original"},
        "USS::Voyager": {"designation": "subsequent"}
    }
},
"classification_sources": {
    "environment": {
        "value": "alpha-quadrant",
        "sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
    },
    "variables": {},
    "classes": {
        "emotion": {
            "puppetlabs.classifier/sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"],
            "importance": {"value": "secondary","sources": ["node"]}
        }
    },
    "logic": {
        "puppetlabs.classifier/sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"],
        "importance": {"value": "primary",}
"sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
},
"config_data": {
  "USS::Enterprise": {
    "designation": {
      "value": "original",
      "sources": ["00000000-0000-4000-8000-000000000000"]
    }
  },
  "USS::Voyager": {
    "designation": {
      "value": "subsequent",
      "sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
    }
  }
}
}
}

This example shows a response that resulted from conflicts:

{
  "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"},
      "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",
                        ...
                    },
                    "defined_by": {
                        "name": "Vulcans",
                        "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
                        ...
                    }
                }
            ]
        },
        "emotion": {
            "importance": [ 
                {
                    "value": "ignored",
                    "from": {
                        "name": "Vulcans",
                        "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
                        ...
                    },
                    "defined_by": {
                        "name": "Vulcans",
                        "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
                        ...
                    }
                },
                {
                    "value": "primary",
                    "from": {
                        "name": "Humans",
                        "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
                        ...
                    },
                    "defined_by": {
                        "name": "Humans",
                        "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
                        ...
                    }
                }
            ]
        }
    }
}
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 endpoint 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.

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": "84b19b51-6db5-4897-9409-a4a3a94b7f09", "name": "Test", "environment": "test"}]}, {"name": "bar", "groups": [{"id": "84b19b51-6db5-4897-9409-a4a3a94b7f09", "name": "Test", "environment": "test"}]}}
```
Assuming token is set as an environment variable, the example below unpins "host1.example" and "host2.example":

curl -k -H "X-Authentication:$TOKEN" \
   -H "Content-Type: application/json" \
   https://<DNS NAME OF CONSOLE>:4433/classifier-api/v1/commands/unpin-from-all \
   -X POST \
   -d '{"nodes": ["host1.example", "host2.example"]}'

The classifier responds with information about each group it is removed from:

```
{"nodes": [
  {
    "name": "host1.example",
    "groups": [
      {
        "id": "2d83d860-19b4-4f7b-8b70-e5ee4d8646db",
        "name": "test",
        "environment": "production"
      }
    ]
  },
  {
    "name": "host2.example",
    "groups": [
      {
        "id": "2d83d860-19b4-4f7b-8b70-e5ee4d8646db",
        "name": "test",
        "environment": "production"
      }
    ]
  }
]}
```

Environments endpoint

Use the environments endpoint to retrieve information about environments in the node classifier. The output will either tell you 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 this endpoint should not be used 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.

© 2020 Puppet, Inc.
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.

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>Controls the maximum number of nodes returned. limit=10 returns 10 nodes.</td>
</tr>
<tr>
<td>offset</td>
<td>Specifies how many nodes to skip before the first returned node. offset=20 skips the first 20 nodes.</td>
</tr>
</tbody>
</table>

CAUTION: In deployments with large numbers of nodes, a large or unspecified limit may 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 since 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:

```
["and", 
["=>", 
["fact", "pressure hulls"], "1"], 
[="", 
["fact", "warp cores"], "0"], 
[">", 
["fact", "docking ports"], "10"]
```

The following node checks into the classifier:

```
{
  "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:

```
{
  "value": true,
  "form": [
    "and",
    {
      "value": true,
      "form": ["=>",{"path": ["fact", "pressure hulls"], "value": "3"}, "1"]
    },
    {
      "value": true,
      "form": ["=",{"path": ["fact", "warp cores"], "value": "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": {
                        "and": [
                            {
                                "value": true,
                                "form": {
                                    "\[\geq\]", {
                                        "path": ["fact", "pressure hulls"],
                                        "value": "3"
                                    }
                                }
                            },
                            {
                                "value": true,
                                "form": {
                                    "\[\leq\]", {
                                        "path": ["fact", "warp cores"],
                                        "value": "0"
                                    }
                                }
                            },
                            {
                                "value": true,
                                "form": {"\[\geq\]" {
                                    "path": ["fact", "docking ports"],
                                    "value": "18"
                                }
                            }
                        ]
                    }
                }
            }
        },
        "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.

© 2020 Puppet, Inc.
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 will be available for the node group to set, and will be the environment that nodes classified into this node group will 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 will override 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 will be 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>
</tbody>
</table>
### Key Definition

**deleted**

An object similar to 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 will not be 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).

**variables**

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).

**children**

A JSON array of the group's immediate children. Children of children are included to the optionally-specified depth.

**immediate_child_count**

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.

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,
```
"rule": ["and", ["="], ["fact", "foo"], "bar"], ["or", ["="], ["name", "db"], ["<"], ["fact", "processorcount"], ["9"], ["="], ["fact", "operatingsystem"], "Ubuntu"]],
"variables": {},
"environment": "test",
"classes": {},
"children": [],
"immediate_child_count": 0
},
{
"name": "grandchild-2",
"id": "71905c11-5295-41cf-a143-31b278cfc859",
"parent": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
"environment_trumps": false,
"rule": ["and", ["="], ["fact", "foo"], "bar"], ["not", ["="], ["fact", "kernel"], "SunOS"]],
"variables": {},
"environment": "test",
"classes": {},
"children": [],
"immediate_child_count": 0
},
"immediate_child_count": 2
},
{
"name": "grandchild-5",
"id": "0bb94f26-2955-4adc-8460-f5ce244d5118",
"parent": "0960f75e-cdd0-4966-96f6-5e60948a7217",
"environment_trumps": false,
"rule": ["and", ["="], ["fact", "foo"], "bar"], ["and", ["<"], ["fact", "processorcount"], ["16"], [">="], ["fact", "kernelmajversion"], ["2"]]],
"variables": {},
"environment": "test",
"classes": {},
"children": [],
"immediate_child_count": 0
}

Permissions

The response returned will vary 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 should contain a rule condition as it would appear in the rule field of a group object. See the documentation on rule grammar for a complete description of how these conditions can be structured.

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 will return a 422 Unprocessable Entity response containing the usual JSON error object. The error object will have a kind of "untranslatable-rule", a msg that describes why the rule cannot be translated, and contain the received rule in details.

If the request does not contain a valid rule, the server will return a 400 Bad Request response with the usual JSON error object. If the rule was not valid JSON, the error's kind will be "malformed-request", the msg will state that the request's body could not be parsed as JSON, and the details will contain the request's body as received by the server.

If the rule does not conform to the rule grammar, the kind key will be "schema-violation, and the details key will be an object with submitted, schema, and error keys which respectively describe the submitted object, the schema that object should conform to, and how the submitted object failed to conform to the schema.

**Related information**

GET /v1/groups on page 406
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 will be 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 will be returned.

**Error responses**

If any of the node groups in the array are malformed, a 400 Bad Request response will be returned. The response will contain the usual JSON error payload. The kind key will be "schema-violation"; the msg key will contain a short description of the problems with the malformed node groups; and the details key will contain 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 will be returned. The response contains the usual JSON error payload, where the kind key will be "inheritance-cycle", the msg key will contain the names of the node groups in the cycle, and the details key will contain 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 will be returned. The response contains the usual JSON error payload, where the kind key will be "unreachable-groups", the msg will list the names of the unreachable node groups, and the details key will contain an array of the unreachable node group objects.

**Related information**

- [Groups endpoint](#) on page 406
- The groups endpoint is used to create, read, update, and delete groups.
- [Node classifier errors](#) on page 450
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 Puppet master.

GET /v1/last-class-update
Use the /v1/last-class-update endpoint to retrieve the time that classes were last updated from the Puppet master.

**Response**
The response will always be an object with one field, last_update. If there has been an update, the value of last_update field will be the time of the last update in ISO8601 format. If the node classifier has never updated from Puppet, the field will be null.

**Update classes endpoint**
Use update classes endpoint to trigger the node classifier to update class and environment definitions from the Puppet master.

POST /v1/update-classes
Use the /v1/update-classes endpoint to trigger the node classifier to update class and environment definitions from the Puppet master. 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:

```bash
--cert <PATH TO CERT>
--key <PATH TO KEY>
--cacert <PATH TO PUPPET CA CERT>
```

**Response**
For a successful update, the service returns a 201 response with an empty body.

**Error responses**
If the Puppet master 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>
</tbody>
</table>
### Key Definition

<table>
<thead>
<tr>
<th>Key</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<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 Puppet master</td>
</tr>
</tbody>
</table>

**Related information**

Enable or disable cached data when updating classes on page 239

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 should contain 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) will be 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 <code>false</code> will be 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 will have no description and this key will not 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 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.

**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.

Responses and keys returned for create group requests depend on the type of error.

**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 should 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>
| 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:

<table>
<thead>
<tr>
<th><strong>Key</strong></th>
<th><strong>Definition</strong></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 caused the constraint to be violated, along with their values.</td>
</tr>
<tr>
<td>details</td>
<td>An object that contains two keys:</td>
</tr>
<tr>
<td></td>
<td>• <strong>conflict</strong>: An object whose keys are the fields of the node group that violated the constraint and whose values are the corresponding field values.</td>
</tr>
<tr>
<td></td>
<td>• <strong>constraintName</strong>: The name of the database 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 the following keys:

<table>
<thead>
<tr>
<th><strong>Key</strong></th>
<th><strong>Definition</strong></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>• <strong>kind</strong>: &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>• <strong>missing</strong>: The name of the missing class or class parameter.</td>
</tr>
<tr>
<td></td>
<td>• <strong>environment</strong>: 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>• <strong>group</strong>: The name of the node group where the error was encountered. Note that, due to inheritance, this might not be the group where the parameter was defined.</td>
</tr>
<tr>
<td></td>
<td>• <strong>defined_by</strong>: 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><strong>Key</strong></th>
<th><strong>Definition</strong></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>
</tbody>
</table>
inherence-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 should be 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 will vary between kinds of errors but will be the same for any given error kind.</td>
</tr>
</tbody>
</table>

**Internal server errors**

An 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 should 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.

Managing applications

Application orchestration provides Puppet language extensions and command-line tools to help you configure and manage multi-service and multi-node applications.

CAUTION: Application orchestration is deprecated and will be removed in a future release.

• Application orchestration on page 451
Create and manage multi-service and multi-node applications.

• Deploying applications with Puppet Application Orchestration: workflow on page 453
Although Puppet Application Orchestration can help you manage any distributed set of infrastructure, it's primarily designed to configure an application stack. The simple application stack used in the following extended example comprises a database server on one machine and a web server that connects to the database on another machine.

• Creating application definitions on page 458
When you create an applications using the Puppet language, you model it in an application definition. An application definition consists of two parts: application components and service resources.

• Declaring application instances on page 461
Declaring an application instance means you have instantiated the application you modeled and have assigned the application's components to nodes and added resource mapping statements.

• Producing and consuming service resources on page 464
When you compose applications, application components can share information with each other by exporting and consuming environment-wide service resources.

• Writing custom service resource types on page 468
When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

• Availability tests on page 469
Availability tests are custom resource providers for testing service resources when you configure applications. They can be used to ensure your nodes are correctly configured and services are running before Puppet configures dependent services on another node.

Application orchestration

Create and manage multi-service and multi-node applications.

CAUTION:

2019.0 is the last short term support (STS) release to support Application Orchestration. While 2019.0 remains supported, will continue to address security issues for Application Orchestration. Feature development has been discontinued. Future releases of will not include Application Orchestration. For more information, see the Puppet Enterprise support lifecycle.

Note: Application orchestration is PE-only feature: While the application orchestration language features work in open source Puppet, the language features alone don't result in ordered runs that respect node dependencies. The Puppet orchestrator command-line tool and service control ordered runs, but is currently available only in Puppet Enterprise. Open source Puppet users can use the application orchestration language features, but open source Puppet does not provide any tools for controlling ordered runs.

Specifically, application orchestration provides:
• Extensions of the Puppet language for describing configuration relationships between components of a distributed application.

• A service that orchestrates configuration enforcement in a directed manner from the node level to the environment level.

• A command-line tool for executing orchestration jobs, inspecting those jobs, and understanding application instances declared in an environment.

When you use application orchestration capabilities:

• Configuration updates are no longer randomly applied when agents choose to check in. Instead, you can directly apply node configurations in a specified order.

• The additions to the Puppet language enable you to model how the logical components of a node configurations relate to each other.

• The orchestrator command-line tool makes it simple to update these configurations in the appropriate order, no matter how many nodes you’re configuring.

In a nutshell, when using application orchestration, you:

1. Use the Puppet language to model the desired state of your application and its supporting infrastructure, describing how information is shared between components.

2. Assign application components to agent nodes.

3. Use the orchestrator to run Puppet in the correct order on the nodes that host application components.

Model a WordPress instance

The Puppet wordpress_app is a module that demonstrates an example application model.

The wordpress_app module contains application components you can use to set up a WordPress database, a PHP application server, and an HAProxy load balancer. With these components, you can build two WordPress applications: a simple LAMP stack or a complex stack that uses load-balancing.

Before you begin, check the module’s dependencies and compatibility. For r10k users, the module contains a Puppetfile that installs the needed dependencies. If you download the module with the puppet module command, the dependencies are installed for you.

Language extensions for application orchestration

One of the main features of application orchestration is its extension of the Puppet language to provide new constructs for managing multi-service and multi-node dependencies. These language extensions include, the application definition, application components, and service resources.

Consider the following example:

A common use-case for Puppet users is the three-tier stack application infrastructure: the load-balancer, the application/web server, and the database server. When you configure these servers, you want the application server to know where the database service is and how they connect so that you can cleanly bring up the application. You then want the load balancer to automatically configure itself to balance demand on a number of application servers. When you update the configuration of these machines, or roll out a new application release, you want the three tiers to reconfigure in the correct order.

Previously, you’d write some Puppet code for each tier of this stack, and classify the nodes responsible for each tier. Then you’d manually run Puppet on each node in the desired order, or wait for configuration to converge after several node runs. In some cases, you even used exported resources to exchange information between the application and database servers, or you pre-filled information in Hiera.

In other words—you did a lot of labor to realize your stack.

With application orchestration functionality, you model each tier as a component of a larger construct—the application—and use Puppet Enterprise to trigger the nodes’ configurations in the order you specify. When you specify the order of the configurations, information is shared between nodes (for example, services are stopped or started) to ensure each component of the stack is created in the correct order.

© 2020 Puppet, Inc.
The application definition

The application definition is a lot like a defined resource type, except that instead of defining a chunk of reusable configuration that applies to a single node, the application definition operates across multiple nodes. The components you declare inside an application can be individually assigned to separate nodes that you manage with Puppet.

The application definition defines the configuration of your application infrastructure---what dependencies nodes have on each other and what information they exchange about those dependencies.

Application components

An application component is an independent bit of Puppet code that can be used alongside one or more other components to create an application. Components are commonly defined types that consist of traditional Puppet resources that describe the configuration of the component (for example, a file, package, or service). But components can be classes or native resources too.

A type, class, or resource becomes an application component when you declare it in an application manifest while producing, consuming, or requiring a service resource (using the export, consume, or require metaparameters).

Service resources

When you compose applications, application components can share information with each other by exporting and consuming environment-wide service resources. This helps components work together even if they are on different servers. An environment service resource works like an exported resource, providing data for other parts of the application to consume. It also helps application components express dependent relationships.

You can also write custom service resource types.

Related information

Writing custom service resource types on page 468
When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

Disable application orchestrator

Turn off application orchestration and stop its commands from running on your system.

1. In the console, click Classification, and from the node group PE Infrastructure, select PE Orchestrator.
2. On the Configuration tab, find the puppet_enterprise::profile::orchestrator class. Select the use_application_services parameter and edit its value to false.

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet_enterprise::profile::orchestrator</td>
<td>use_application_services</td>
<td>false</td>
</tr>
</tbody>
</table>

3. Click Add parameter and commit the change.
4. Set up a job and run Puppet on the PE Infrastructure node group to enforce your changes.

Deploying applications with Puppet Application Orchestration: workflow

Although Puppet Application Orchestration can help you manage any distributed set of infrastructure, it’s primarily designed to configure an application stack. The simple application stack used in the following extended example comprises a database server on one machine and a web server that connects to the database on another machine.

With previous Puppet coding techniques, you’d write classes and defined types to define the configuration for these services, and you’d pass in class parameter data to tell the web server class how to connect to the database. With application orchestration, you can write Puppet code so this information can be exchanged automatically. And when you run Puppet, the services will be configured in the correct order, rather than repeatedly until convergence.

© 2020 Puppet, Inc.
Application orchestration workflow

The application orchestration workflow illustrates the major steps in the application orchestration workflow—from authoring your application to configuring it with the orchestrator.

Before you begin

Prior hands-on experience writing Puppet code is required to author applications for use with application orchestration. You should also be familiar with modules.

Details about the code for this LAMP application module are available at puppetlabs/appmgmt-module-lamp.
1. Create the service resources and application components.
   In the applications you compose, application components share information with each other by exporting and consuming environment-wide service resources.

   An application component is an independent bit of Puppet code that can be used alongside one or more other components to create an application. Components are often defined types that consist of traditional Puppet resources that describe the configuration of the component (file, package, and service, for example).

   a) Create the Sql service resource in lamp/lib/puppet/type/sql.rb.

```ruby
Puppet::Type.newtype :sql, :is_capability => true do
  newparam :name, :namevar => true
  newparam :user
  newparam :password
  newparam :port
  newparam :host
end
```

   b) Define the database application component in lamp/manifests/db.pp.

```ruby
# Creates and manages a database
define lamp::db(
  $db_user,
  $db_password,
  $host = $::fqdn,
  $port = 3306,
  $database = $name,
) {

  include mysql::bindings::php

  mysql::db { $name:
    user     => $db_user,
    password => $db_password,
  }
}
```


```ruby
define lamp::web(
  $port = '80',
  $docroot = '/var/www/html',
) {

  class { 'apache':
    default_mods => false,
    default_vhost => false,
  }

  apache::vhost { $name:
    port    => $port,
    docroot => $docroot,
  }
}
```

   d) Define the application server component in lamp/manifests/app.pp.

```ruby
# Creates and manages an app server
define lamp::app (
  $docroot,
  $db_name,
  $db_port,
  $db_user,
  $db_host,
) {

Lamp::Web produces Http {
  ip     => $::ipaddress,
  port   => $port,
  host   => $::fqdn
}
```
2. Create the application definition.

The application definition (or model) is where you connect all the pieces together. It describes the relationship between the application components and the exchanged service resources.

Since the application definition shares the name of the module, you put it in `lamp/manifests/init.pp`.

```perl
application lamp (  
  $db_user,  
  $db_password,  
  $docroot = '/var/www/html',  
) {  
  lamp::web { $name:  
    docroot => $docroot,  
    export => Http"lamp-$name",  
  }  
  lamp::app { $name:  
    docroot => $docroot,  
    consume => Sql"lamp-$name",  
  }  
  lamp::db { $name:  
    db_user     => $db_user,  
    db_password => $db_password,  
    export      => Sql"lamp-$name",  
  }  
}
```

3. Instantiate the application.

In the application instance, create a unique version of your application and specify which nodes to use for each component.

```perl
template::file (  
  '/etc/puppetlabs/codegen/manifests/site.pp',  
  content => <<code
site {  
  lamp { 'stack':  
    db_user => 'example',  
    db_password => 'lightyear',  
    nodes => {  
      Node['1234.rgbank.net'] => [Lamp::Web['stack'], Lamp::App['stack']],  
      Node['5678.rgbank.net'] => Lamp::Db['stack'],  
    }  
  }  
}
code
)  
```
4. Use the orchestrator commands to run Puppet and configure the application.
   a) Run `puppet app show` to see the details of your application instance.
   b) Run `puppet job run` to run Puppet across all the nodes in the order specified in your application instance.
   c) Run `puppet job list` to show running and completed orchestrator jobs.

   At the start of a job run, the orchestrator prints a job plan that shows what’s included in the run and the expected node run priority. The nodes are grouped by depth. Nodes in level 0 have no dependencies and will run first. Nodes in the levels below are dependent on nodes in higher levels.

   As your job progresses, the orchestrator will print results of each node run after it completes. A "Success!" message prints when all jobs complete.

   Related information

   Declaring application instances on page 461
   Declaring an application instance means you have instantiated the application you modeled and have assigned the application's components to nodes and added resource mapping statements.

   Application definitions on page 458
   The application definition is a lot like a defined resource type except that instead of defining a chunk of reusable configuration that applies to a single node, the application definition operates at a higher level. The components you declare inside an application can be individually assigned to separate nodes you manage with Puppet.

   Producing and consuming service resources on page 464
   When you compose applications, application components can share information with each other by exporting and consuming environment-wide service resources.

   Running jobs on the command line
Creating application definitions

When you create an applications using the Puppet language, you model it in an application definition. An application definition consists of two parts: application components and service resources.

After you’ve created your application definition, your application can be instantiated any number of times, as discussed in declaring application instances.

Application definitions

The application definition is a lot like a defined resource type except that instead of defining a chunk of reusable configuration that applies to a single node, the application definition operates at a higher level. The components you declare inside an application can be individually assigned to separate nodes you manage with Puppet.

For example, if you want to manage a simple LAMP stack, you would create an application definition that contains two components. A component for the Apache web server with your PHP application and one for the MySQL database service. You could make it fancier with multiple Apache components, a load balancer component and even a separate component for your application but our example will keep it simple.

Where class and defined resource type definitions are for modeling a collection of node configuration, the application definition is for modeling configuration at a higher-level—site or environment wide.

Within an application definition, you declare application components. These are declarations of classes or resources that express a relationship to environment-wide services. A common pattern is a three-tier application stack where a load balancer relies on one or more application servers, which rely on one or more database servers.

The application definition, therefore, describes the relationship between the application components and the exchanged service resources.

Applications definitions exist in module manifests:

- Applications definitions are located in module manifests and follow standard autoloader behavior. For example, an application directory uses the typical path: `<ENVIRONMENT DIRECTORY>/modules/<MODULE NAME>/manifests/init.pp`, where `init.pp` would define the application `MODULE NAME`.
- Any application component (classes, resources, or defined types) should be in its own file in the module’s manifests directory, and each file should have the `.pp` file extension.
- Service resources live in a module's `type` directory and uses the typical path `<ENVIRONMENT DIRECTORY>/modules/<MODULE NAME>/lib/puppet/type/<SERVICE RESOURCE.rb>`.

Tip: The module fundamentals documentation explains the standard module layout.

Application components in application definitions

An application component is an independent bit of Puppet code that can be used alongside one or more other components to create an application. Components are commonly defined types that consist of traditional Puppet resources describing the configuration of the component (its files, packages, and services) but can be classes or native resources too.

A type, class, or resource becomes an application component when you declare it in an application manifest while producing, consuming, or requiring a service resource (using the `export`, `consume`, or `require` statements).

Here is an example of an application component definition, in this case, a mock MySQL database:

```puppet
# Creates and manages a database
define lamp::db ( $db_user, $db_password, $host = ::fqdn, $port = 3306, $database = $name, ) {
    notify { "Hello! This is the ${name}'s lamp::db component" }
}
```

© 2020 Puppet, Inc.
include mysql::bindings::php

mysql::db { $name:
  user => $db_user,
  password => $db_password,
}
...
}

Lamp::Db produces Sql {
  host => $host,
  port => $port,
  database => $database,
  user => $db_user,
  password => $db_password,
}

The following application component is an app server that relies on the previously defined database.

# Creates and manages an app server

define lamp::app (
  $docroot,
  $db_name,
  $db_port,
  $db_user,
  $db_host,
  $db_password,
  $host = $::fqdn,
) {
  notify { "Hello! This is the ${name}'s rgbank::web component" }
  ...
}

Lamp::App consumes Sql {
  db_host => $host,
  db_port => $port,
  db_name => $database,
  db_user => $user,
  db_password => $password,
}

Finally, we have a webserver as well.

define lamp::web(
  $port = '80',
  $docroot = '/var/www/html',
) {
  class 'apache':
    default_mods => false,
    default_vhost => false,
  }
  apache::vhost { $name:
    port => $port,
    docroot => $docroot,
  }
Combining application components in an application definition

The *application definition* groups components together and defines the flow of data between them. Most applications rely on some combination of components.

The application definition requires the *application* keyword and then one or more components.

In the following example, we've grouped the `lamp::db` component, the `lamp::web` component, and the `lamp::app` component in an application definition called `lamp`. Remember, the `lamp::db` component *exports* (produces) the `Sql` service resource, which is then *consumed* by the `lamp::app` component. The `lamp::web` component produces the `Http` service resource.

*Note:* For details on service resource mapping statements, which are part of the application instance declaration, refer to Adding service resource mapping statements. For more information about service resource types, refer to Producing and consuming service resources and Writing custom service resource types.

```perl
application lamp {
    $db_user,
    $db_password,
    $docroot = '/var/www/html',
}

lamp::web { $name:
    docroot => $docroot,
    export => Http[lamp-$name],
}

lamp::app { $name:
    docroot => $docroot,
    consume => Sql[lamp-$name],
}

lamp::db { $name:
    db_user => $db_user,
    db_password => $db_password,
    export => Sql[lamp-$name],
}
```

**Related information**

Producing and consuming service resources on page 464

When you compose applications, application components can share information with each other by exporting and consuming environment-wide *service resources*.

Writing custom service resource types on page 468

When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

Adding service resource mapping statements on page 463

© 2020 Puppet, Inc.
Service resource mapping statements are the final piece of the application declaration. Add service resource mapping statements either in your site manifest (`site.pp`) or in modules that contain service resources.

---

**Declaring application instances**

Declaring an application instance means you have instantiated the application you modeled and have assigned the application's components to nodes and added resource mapping statements.

An application can be instantiated any number of times with a variety of component-to-node assignment combinations—from all components in an application assigned to one node, to the same components spread across several nodes in your infrastructure.

---

**Declaring application instances in site.pp**

You declare application instances in an environment's main manifest (`site.pp`).

Application instances go inside a `site { }` block in the environment's main manifest, which allows applications to be evaluated separately from node-level resources. You have one `site { }` per environment, and you declare all application instances for an environment within it.

Within the `site { }` block, applications are declared like defined types. They can be declared any number of times, but their type and title combination must be unique within an environment.

The following section provides some example application instance declarations.

---

**Example application instance declarations**

The following example shows how you’d declare an instance of an application with two or more components. In this case, `database.vm` is hosting the database while `appserver.vm` is hosting the app server.

**Note:** In the following examples, the `$codedir` variable refers to Puppet’s code and data directory.

Component titles are taken from application definitions. In these examples, we refer to them as `<COMPONENT TITLE>`.

In addition, application instances cannot use static values for parameters. In these examples, we use `<USER NAME>` and `<PASSWORD>`.

```
$codedir/environments/production/manifests/site.pp

site {
  lamp { 'stack':
    db_user      => <USER NAME>,
    db_password  => <PASSWORD>,
    nodes        => {
      Node['database.vm'] => Lamp::Db[<COMPONENT TITLE>],
      Node['appserver.vm'] => Lamp::App[<COMPONENT TITLE>],
    },
  }
}
```

Alternatively, you can have a single node application with multiple components on that node, as shown in the following application declaration example. In this case, `example.vm` is hosting both the MySQL database and the WordPress application.

```
$codedir/environments/production/manifests/site.pp

site {
  lamp { 'dev':
    db_user      => <USER NAME>,
    db_password  => <PASSWORD>,
    nodes        => {
    }
  }
}
```
Node assignment syntax for application declarations

A crucial step in declaring application instances is assigning application components to nodes. Node mapping is accomplished with the nodes attributes in the application instance declaration in site.pp.

In this example, example01.node.cert is assigned two components, and example02.node.cert is assigned one.

```
      nodes => {
        Node['example01.node.cert'] => [ Component['one'], Component['two'] ],
        Node['example02.node.cert'] => Component['three'],
      },
```

The general form of the node mapping statement is structured as follows:

- The nodes attribute.
- Node followed by brackets ([) that contain the [certname of the node you're mapping the component to.
- A => (called an arrow, “fat comma,” or “hash rocket”).
- An opening bracket ([) followed by a space, followed by an array of components, with each component followed by brackets ([ ]) containing the component title, and then an additional space. Do not put a space between the component and the component's name. The title of the component declared in the application instance must match the title of the component given in the application manifest. (The defined type $name variable is useful in the application definition but cannot be used in application instances.)
- A closing (]) bracket.
- A comma.

**Important:** If you do not map your components to nodes, they will not show up in the environment graph and therefore not be usable with Puppet Application Orchestration.

One common error to avoid when assigning application components to nodes is not to map a single component to multiple nodes.

If you attempt to map the same component to multiple nodes, as shown in the following example, your application will not be properly configured.

```
lamp { 'example_instance':
    wp_db_user      => <USER NAME>,
    wp_db_password  => '<PASSWORD>',
    nodes           => {
        Node['database.vm'] => [ Lamp::Db[<COMPONENT TITLE>],
        Node['example.vm']  => [ Lamp::Db[<COMPONENT TITLE>],Lamp::Web[<COMPONENT TITLE>]],
    }
}
```

Note that you cannot use a variable (such as $name) to declare the components. Instead, follow this example to map the components to the nodes:

```
elk { 'split':
    nodes => {
        Node['centos7-1.vm'] => [ Elk::Es['split'] ],
        Node['centos7-2.vm'] => [ Elk::Kibana['split'] ]
    }
}
```
**Related information**

**Writing custom service resource types** on page 468
When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

**Producing and consuming service resources** on page 464
When you compose applications, application components can share information with each other by exporting and consuming environment-wide **service resources**.

**Adding service resource mapping statements**

Service resource mapping statements are the final piece of the application declaration. Add service resource mapping statements either in your site manifest (**site.pp**) or in modules that contain service resources.

**In the site manifest**

Declaring service resource mapping statements in your site manifest (**site.pp**) allows you to use modules (such as those from the Forge) that are not wired for application management alongside instantiated applications.

```
$codedir/environments/production/manifests/site.pp

site {
  lamp { 'example_instance':
    wp_db_user => '<USER NAME>',
    wp_db_password => '<PASSWORD>',
    nodes => {
      Node['database.vm'] => [Lamp::Db[<COMPONENT TITLE>]],
      Node['appserver.vm'] => [Lamp::Web[<COMPONENT TITLE>]]
    }
  }

Lamp::Db produces Sql {
  user => $user,
  password => $password,
  host => pick($::mysql_host_override, $::fqdn),
  database => $dbname,
  type => 'mysql',
}
```

**In modules with service resources**

You can add service resource mappings to modules that include service resources. Declare the mappings in the manifest that conforms to autoloader rules, typically outside of the class/defined-resource type.

```
$modulepath/mysql/manifests/db.pp

define lamp::db {
  ....
}

Lamp::Db produces Sql {
  user => $user,
  password => $password,
  host => pick($::mysql_host_override, $::fqdn),
  #port => not used here, will default as described in the definition
  database => $dbname,
  type => 'mysql',
}
```
Producing and consuming service resources

When you compose applications, application components can share information with each other by exporting and consuming environment-wide service resources.

Exporting and consuming service resources

Service resources help components work together even if they are managed by Puppet on different servers.

An environment service resource works like an exported resource, providing data for other parts of the application to consume but also helps application components express inter-dependent relationships.

Specifically, service resources:

- Transport data between nodes participating in an application.
- Provide an abstraction so that you no longer need to hard-code dependencies in modules.
- Provide service resource availability tests at the node level to understand when a component is “ready.” Service readiness can tell the Puppet agent to pause execution until the required service is available.
- Live in a module’s type directory and uses the typical path <ENVIRONMENT DIRECTORY>/modules/<MODULE NAME>/lib/puppet/type/<SERVICE RESOURCE.rb>.

Essentially, service resources serve as checkpoints that the Puppet master uses to determine when it is safe to trigger a Puppet run on a node that consumes a resource.

Service resource instances must be unique across an environment. Like other kinds of resource uniqueness, this means they must have a unique type and title. Also, just like exported resources, service resources can be located using resource collectors.

Service resources are stored, like other resources, in PuppetDB. They are subject to the lifecycle of any resource and the node whose catalog they are in. In addition, service resources adhere to the standard PuppetDB rules for purging nodes and resources.

Producing a service resource with export

When declaring an application component, the export statement is used to produce an instance of the service resource as produced by the node managing that application component.

In the following example, the mysql::db application component exports the Sql service resource. (The values of the parameters for the exported resource are defined in the component.)

```
mysql::db { $name:
    user       => $wp_db_user,
    password   => $wp_db_password,
    export     => Sql[$name],
}
```

Syntax

'export' '=>' CapabilityType[Name]

The general form of the export statement is structured like an attribute/value pair:

- The export keyword.
- A => (called an arrow, “fat comma,” or “hash rocket”).
- The name of the capability type, followed by a pair of brackets ([ ]) that contain the name to be given the service resource.

The CapabilityType must be the name of a service type for which there is a produces declaration for the type of resource being instantiated.
The resulting service resource will have the name that is specified in the `export` reference; the remaining attributes of the service resource will be filled by evaluating the corresponding `produces` declaration.

The produced service resource must be unique, by type and title, within the current environment.

**Consuming a service resource with consume**

When instantiating a resource that is declared to consume a capability, the `consume` statement can be used to fill attributes of the resource being instantiated.

In the following example, the values of the `wordpress::instance::app` are assigned when the component consumes the `Sql` service resource.

```ruby
wordpress::instance::app { $name:
  install_dir => "/var/www/${name}",
  consume => Sql[$name],
}
```

**Syntax**

'`consume`' => CapabilityType[Name]

The general form of the `consume` statement is structured like an attribute/value pair:

- The `consume` keyword.
- A `=>` (called an arrow, “fat comma,” or “hash rocket”).
- The name of the capability type, followed by a pair of brackets ([ ]) that contain the name to be given the service resource.

The reference `CapabilityType[Name]` must specify a service resource that is produced on a node, including the current node, in the current environment.

**Using require between components**

Requiring a resource declares a dependency between a component and a service resource.

If you need to depend on a service resource but don't need to consume its information, you can use the `require` metaparameter, which behaves as it does elsewhere in Puppet.

**Related information**

Availability tests on page 469

Availability tests are custom resource providers for testing service resources when you configure applications. They can be used to ensure your nodes are correctly configured and services are running before Puppet configures dependent services on another node.

**Mapping information from components to service resources**

You use the `produces` and `consumes` statements to map information between components and service resources.

**Mapping with the produces statement**

The `produces` statement allows you to define a mapping from a component providing a service to a service resource. Further, it allows you to define how a service resource is created when an application is configured, and it defines the complete content of the service resource.

When you produce a service resource, you have access to all the top-level variables and facts available on the source node, as well as the properties of the source component. As noted below, function calls are evaluated only during Puppet runs that produce the source node's catalog.

In this example, the `mysql::db` defined type `produces` the `Sql` resource capability.

```ruby
Mysql::Db produces Sql {
  user => $user,
  password => $password,
}```
Let's examine some of the use-cases in this statement:

- To capture information about the component, its properties can be accessed directly. In this example `user`, `password`, and `database` of the `Sql` service resource is set directly from the `Mysql::Db` user, `password`, and `database` properties.

- To capture information about the node hosting the component, facts describe the state at the time of configuration. In this example the `host` of the `Sql` service resource is set to the fully qualified hostname (`$::fqdn`) of the node running the database.

- To capture information about the node's environment, top-level variables, set by the node classifier, can be accessed. In this example the `$::mysql_host_override` could be set through the node classifier to define the DNS name to use for accessing databases on a specific node.

- Constant values can be used to supply additional information. In this example we specify the capability's `type` as `mysql` to allow its consumers to choose the correct database connector.

- Function calls allow access to a significant part of Puppet's power, from simple `lowercase()` to complex `hiera()` lookups. Note that function calls---like everything else here---are only evaluated during the Puppet run when the producing node's catalog is compiled. In this example the `pick()` function is used to choose between the node classifier supplied value and the fact value, using the latter as the default.

**Note:** Since allowing access to a component's properties allows full access to its scope, referencing non-top-level values is technically possible (through default values and function calls) but should not be done in `produces` statements, as they depend on the evaluation order during catalog compilation, and non-top-level values might not actually be available for evaluation.

**Syntax**

A `produces` statement has the following syntax:

```
ResourceType 'produces' CapabilityType ( '{' (AttributeName '=>' Expression ',')*' '}')?
```

The general form of the `produces` statement contains:

- The name of the resource type producing the service resource.
- The `produces` keyword.
- The name of the service resource created.
- An opening curly brace (`{`).
- Any number of attribute and value pairs, each of which consists of:
  - An attribute name, which is a lowercase word with no quotes.
  - A `=` (called an arrow, “fat comma,” or “hash rocket”).
  - A value, which can have any data type.
  - A trailing comma.
  - A closing curly brace (`}`).

**Behavior**

- The `ResourceType` can be any normal resource type, including classes and defined types.
- The `CapabilityType` must be a service resource type.
- The statement does not result in the creation of a service resource; instead, it serves as the blueprint for creating the service resource using the `export` statement.
• Note that for any pair of (ResourceType, CapabilityType), we can have at most one produces statement. If there is more than one produces statement, Puppet will not know which one to use, and the compiler will flag this as an error.
• The possible names for the AttributeName are the names of the attributes of the CapabilityType. For any attribute with name aname which is not explicitly listed, we act as if the user has written aname => $aname.
• The Expression for each AttributeName is evaluated in a scope that contains top-level variables, including the facts for the node on which the underlying Resource is being instantiated. In addition, the scope contains bindings for all the parameters/attributes from the instantiation of the underlying ResourceType.

Mapping with the consumes statement
The consumes statement allows you to define which components use a resource capability produced by some other component.

In the following example, the Wordpress::Instance::App defined type consumes the Sql resource capability. Here the attributes for the defined type are derived from the attributes of the service resource it’s consuming (for example, the Wordpress::Instance::App db_password parameter setting comes from the $password parameter setting defined in the Sql service resource).

Wordpress::Instance::App consumes Sql {
    # defined_type_attribute => $capability_attribute,
    db_name       => $name,  
    db_host       => $host,  
    db_user       => $user,  
    db_password   => $password,
}

Now these two defined types (Mysql::Db and Wordpress::Instance::App) can exchange data about themselves through the Sql service resource. Service resources make it possible to create interfaces between services (potentially across nodes) from existing code without modification. Further, since they provide a level of abstraction at the service level, you could swap out mysql::db for postgresql::server::db, and the consuming service wouldn’t mind in the least.

Note: Properties specified in a component will override any values contained in a service resource.

Syntax
A consumes statement has the following syntax:

    ResourceType 'consumes' CapabilityType ( '{'  
        (AttributeName '=>' Expression ',')*  
    '}' )

The general form of the consumes statement contains:
• The name of the resource type consuming the service resource.
• The consumes keyword.
• The name of the service resource consumed.
• An opening curly brace ({}).
• Any number of attribute and value pairs, each of which consists of:
  • An attribute name, which is a lowercase word with no quotes.
  • A => (called an arrow, “fat comma,” or “hash rocket”).
  • A value, which can have any data type.
  • A trailing comma.
• A closing curly brace (}).
Behavior

• Similar to produces, the ResourceType must be a normal resource type, and the CapabilityType must be a service resource type. There can be at most one consumes statement for every (ResourceType, CapabilityType) pair.
• The consumes statement alone does not change how an instance of the ResourceType is created. When an instance of the given ResourceType is instantiated and uses the consume statement, the mappings established by the consumes statement are used to fill the attributes/parameters of the resource from the service resource.
• The possible names for the AttributeName are all the names of attributes/parameters that ResourceType accepts. For each attribute/parameter name aname of the ResourceType, the consumes statement will act as if the statement contained a mapping aname => $aname as long as the CapabilityResource has an attribute aname and the attribute/parameter of the instance of ResourceType is not set explicitly in the resource instantiation. Any attribute/parameter name of ResourceType that does not coincide with the name of an attribute of CapabilityType will not be affected by the consumes statement. It is legal to not use some of the attributes of the CapabilityType in the mapping at all.
• The Expression for each AttributeName is evaluated in a scope that contains top-level variables as well as a variable for each attribute of the CapabilityResource, bound to the value of that attribute in the consumed CapabilityResource.

Implicit parameter mappings

If the attribute names of both defined types that need to exchange data are identical, the mapping between them within a service resource is automatic.

Let’s say you have a defined type called mysql::db (that produces Sql) and a defined type called wordpress::app (that consumes Sql), and both share the attribute name, $param.

You can simply express this as:

Mysql::Db produces Sql
Wordpress::App consumes Sql

Writing custom service resource types

When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

A service resource type looks very much like any other custom type. Service resources live in a module’s type directory and uses the typical path <ENVIRONMENT_DIRECTORY>/modules/<MODULE_NAME>/lib/puppet/type/<SERVICE RESOURCE.rb>.

The custom type documentation has all the information you need for writing custom types.

Since Puppet types and providers are written in Ruby, we recommend you have some experience writing Ruby code before writing any custom types.

The fundamentals of writing custom service resource types

A service resource typically adheres to the same format as a standard custom type.

    Puppet::Type.newtype :sql, :is_capability => true do
      newparam :name, :namevar => true
      newparam :user
      newparam :password
      newparam :host
      newparam :port do
        defaultto :standard
        newvalues(:standard, /\d+\$/)  

© 2020 Puppet, Inc.
newparam :database
define my::db(
    $user = 'myuser',
    $password = 'mypasswd',
    $host = 'myhost',
    $port = '5432',
    $database = 'mydb',
    $provider = 'tcp',
) {  
  
}

The format of the service resource type adheres to the fundamental rules of custom types (as outlined in the custom types documentation):

- Types are created by calling the newtype method on the Puppet::Type class.
- The name of the type is the only required argument to newtype. The name must be a Ruby symbol, and the name of the file containing the type must match the type’s name.
- The newtype method also requires a block of code, specified with either curly braces ({{ ... } }) or the do ... end syntax. This code block will implement the type, and contains all of the properties and parameters. The block will not be passed any arguments.
- Every type must have at least one mandatory parameter: the namevar.
- The isCapability property lets you specially mark these as environment wide service resources for extraction and injection into catalogs of the producer or consumer. Otherwise, sql is just a standard resource that has to be declared on a node.
- Parameters are defined essentially exactly the same as properties; the only difference between them is that parameters never result in methods being called on providers. To define a new parameter, call the newparam method. This method takes the name of the parameter (as a symbol) as its argument, as well as a block of code.

### Availability tests

Availability tests are custom resource providers for testing service resources when you configure applications. They can be used to ensure your nodes are correctly configured and services are running before Puppet configures dependent services on another node.

#### Example service resource with built-in availability test

When you create service resources, you want to include availability tests when necessary.

The following example shows two example availability tests. These tests are based on puppet-healthcheck.

- **TCP**
  - The provider is called tcp.
  - This check performs a simple ping against a host and port to determine if the service is up.

- **PostgreSQL**
  - The provider is called pg.
  - This check logs into specified database to do a simple SELECT 1 to determine if the database is up.
  - This check requires that you install the postgres-pr gem on the agent that will run the check.

In the following example, the component (My::Db) produces a service resource (Sql), and passes the tcp provider from the component into the service resource.

```ruby
define my::db(
    $user = 'myuser',
    $password = 'mypasswd',
    $host = 'myhost',
    $port = '5432',
    $database = 'mydb',
    $provider = 'tcp',
) {  
  
}
```
My::Db produces Sql {
  user => $user,
  password => $password,
  host => $host,
  port => $port,
  database => $dbname,
  provider => $provider,
}

Based on this example, the following parameters from the sql type dictate the behavior of the availability test providers:

- **host**
  - The host to run the test on.
  - Default is 127.0.0.1.
- **port**
  - The port to run the test on.
  - Defaults is 80 or 5432, depending on it's a TCP or PSQL test.
- **timeout**
  - How long to run the test.
  - Defaults is 60 seconds.
- **ping_interval**
  - How long to wait between tests.
  - Defaults is 1 second.
- **user**
  - The user to log into the database with.
  - There is no default value.
- **password**
  - The password to log into the database with.
  - There is no default value.
- **database**
  - The database to connect to.
  - There is no default value.

**Writing availability tests**

You can write your own availability tests as needed. To write an availability check, you need to create a custom type and a custom provider.

**Writing the type**

The type needs to be defined as a service resource. The following example shows the basic format of a service resource.

```ruby
Puppet::Type.newtype :mytype, :is_capability => true do
  newparam :type1
  newparam :type2
  newparam :type3
end
```

Here, you add whatever parameters this service resource needs to function.

In the module that defines your application, this service resource type would be located at `lib/puppet/type/<YOUR TYPE>/<YOUR TYPE>.rb`. © 2020 Puppet, Inc.
Writing the provider

The corresponding provider is what performs the availability test. Most parts of the provider are dependent on how you want the test to function. However, there are two required methods:

- `exists?`: The method with which you call your availability test code. This is also where you can pass in any parameter from your type to your availability test method.
- `create`: The method on which you can raise a `Puppet::Error`. This will only be called if the `exists?` method returns `false`.

The following example shows the format of the provider portion of a basic availability test. As noted in the example, the default behavior of the test is to pass all parameters from the service resource to the availability test. You can also pass specific parameters, as shown in the example.

(Remember, the service resource type would be located at `lib/puppet/provider/<YOUR TYPE>/<YOUR TYPE>.rb`.)

```ruby
require 'puppet/provider'

class Puppet::Provider::Tcp < Puppet::Provider
  def my_availability_test
    # availability test code
    Puppet::Info "Doing an availability test"
    true
  end

  # default, passes all parameters from the service resource to the availability test
  # or the commented out code: passing specific parameters to the test
  def exists?
    my_availability_test
    # my_availability_test(resource[:type1], resource[:type2], resource[:type3])
  end

  def create
    raise Puppet::Error, "The availability test failed"
  end
end
```

Ensuring test runs only on consuming node

An availability test will run only on the node consuming the service resource. For example, if you have an API node and a database node, and the database node produces a `Sql` capability resource consumed by the API node, the availability test will only run on the API node after the database node has been configured.

The availability test will also run before any other resources are configured during the Puppet run. If it fails, all dependencies will be skipped. If it successfully makes a connection, the Puppet run will move on and configure the remaining parts of the application.

To ensure the availability test only runs on the consuming node, you must include the following code on your health check method:

```ruby
def my_availability_test
  return true if resource.tags.grep(/^producer:/).any?
  # rest of the availability test code
end
```

Related links

- Custom type and a provider documentation
- Working examples

© 2020 Puppet, Inc.
See the following provider directories in the puppet-healthcheck module for working examples.

Related information
Writing custom service resource types on page 468
When writing custom service resource types, remember that service resources typically adhere to the same format as standard custom types.

Orchestrating Puppet and tasks

Puppet orchestrator is an effective tool for making on-demand changes to your infrastructure.

With orchestrator you can:

• Automate tasks and eliminate manual work across your infrastructure and applications.
• Initiate Puppet runs whenever you need to update agents.
• Group the servers in your network according to immediate business needs. By leveraging orchestrator with PuppetDB, which stores detailed information about your nodes, you can search and filter servers based on metadata. No static lists of hosts, no reliance on complicated host-naming conventions.
• Connect with thousands of hosts at the same time without slowing down your network. The Puppet Communications Protocol (PCP) and message broker efficiently mediate communications on your network even as your operational demands grow.
• Distribute the Puppet agent workload by adding masters that are dedicated to catalog compilation as you increase in scale. Compile masters efficiently process requests and compile code for environments that have thousands of nodes. For more information, see Installing compile masters on page 205.
• Take advantage of the tasks installed with Puppet Enterprise. Use the package task to inspect, install, upgrade, and manage packages or the service task to start, stop, restart, and check the status of services running on your systems. Additional tasks are available from the Puppet Forge.
• Write your own tasks in any programming language that your target nodes will run, such as Bash, PowerShell, or Python.
• Integrate server logging, auditing, and per node role-based access control (RBAC).

Note: If you've been using MCollective to do some of this work, it's time to move to orchestrator. Start by comparing your MCollective workflows to Puppet Enterprise features. For more information, see Move from MCollective to Puppet orchestrator on page 751.
Running jobs with Puppet orchestrator

With the Puppet orchestrator, you can run two types of "jobs": on-demand Puppet runs or Puppet tasks.

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.

Bolt is an open source task runner that automates the manual work that you do to maintain your infrastructure.

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 job to enforce to make those changes at will.

Tasks 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 and puppet-app 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) and the Puppet Communications Protocol (PCP).

- 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 master of masters (MoM) and in the pe-puppetserver service on compile masters. 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 compile masters connect to the orchestrator, and the orchestrator uses the brokers to direct messages to PXP agents connected to the compile masters. When using compile masters, PXP agents running on PE components (the Puppet master, PuppetDB, and the PE console) connect directly to the orchestrator, but all other PXP agents connect to compile masters via load balancers.

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 requests the environment graph from Puppet Server.
5. The orchestrator creates the job ID and starts polling nodes in the job to check their statuses.
6. The orchestrator queries PuppetDB for the agent version on the nodes targeted in the job.
7. The orchestrator tells the PCP broker to start runs on the nodes targeted in the job, and Puppet runs start on those agents.
8. The agent sends its run results to the PCP broker.
9. 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. Since tasks are Puppet code, they must be deployed into an environment on the Puppet master. 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 master 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 might include tuning the pe-orchestration-services JVM heap size, increasing the limit on open file descriptors for pe-orchestration-services, and tuning ARP tables.

© 2020 Puppet, Inc.
• 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 high availability (HA) is configured, the agent will point to additional PCP broker endpoints in the case of failover.

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 compile masters 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 MoM 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 or C:/ProgramData/PuppetLabs/pxp-agent/var/spool (on Windows). Results are kept for 14 days.

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 PE client tools on page 213
PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily managed by Puppet.

Move from MCollective to Puppet orchestrator on page 751
Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

**Orchestrator settings**

Puppet orchestrator has several internal settings to help tune and manage your orchestration service. Most of these do not need to be changed, but can be edited in Hiera if needed.

**Orchestration services settings**

**global.conf**: 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>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.cert.pem</td>
</tr>
<tr>
<td>ssl-key</td>
<td>Private key path for the orchestrator host.</td>
<td>/etc/puppetlabs/orchestration-services/ssl/&lt;orchestrator-host-fqdn&gt;.private_key.pem</td>
</tr>
<tr>
<td>ssl-ca-cert</td>
<td>CA file path</td>
<td>/etc/puppetlabs/puppet/ssl/ca.pem</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)

**bootstrap.cfg**: 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`.

**webserver.conf and web-routes.conf**: The pcp-broker and orchestrator HTTP services

/etc/puppetlabs/orchestration-services/conf.d/webserver.conf describes how and where to the 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>/etc/puppetlabs/orchestration-services/request-logging.xml</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>want or need</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>true</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>8143</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>want or need</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>
</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>agent</td>
<td>Run on agent nodes.</td>
<td>The ability to run on nodes using the console or orchestrator. Instance must always be &quot;*&quot;.</td>
</tr>
</tbody>
</table>
### Assign task permissions to a user role

1. In the console, click **Access control > User roles**.
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, **facte_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 will prompt 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 289
  Puppet Enterprise users generate tokens to authenticate their access to certain PE command-line tools and API endpoints. 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 281
  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 282
  You can mix and match permissions to create custom user roles that provide users with precise levels of access to PE actions.

### Enable cached catalogs for use with the orchestrator (optional)

Enabling cached catalogs on your agents ensures Puppet will not enforce any catalog changes on your agents until you run an orchestrator job to enforce changes.

When you use the orchestrator to enforce change in a Puppet environment (for example, in the production environment), you want the agents in that environment to rely on their cached catalogs until you run an orchestrator job that includes configuration changes for those agents. Agents in such environments will check in during the run interval (30 minutes by default) to reinforce configuration in their cached catalogs and only apply new configuration when you run Puppet with an orchestration job.

---

### Table: Task Permissions and Definitions

<table>
<thead>
<tr>
<th>Type</th>
<th>Permission</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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</td>
<td>The ability to view node data imported from . 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 will not appear if no rules have been specified for it.
**Note:** This is an optional configuration. You can run Puppet on nodes with the orchestrator in workflows that don't require cached catalogs.

1. Run Puppet on the new agents.

   **Important:** Be sure you run Puppet on the new agents **before** assigning any application components to them or performing the next step.

2. In each agent's `puppet.conf` file, in the `[agent]` section, add `use_cached_catalog=true`. To complete this step, choose one of the following methods:
   - From the command line on each agent machine, run the following command:
     ```
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`. This adds the setting to **all** agents in that environment.
     ```
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.

## 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 Puppet master 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 Puppet master 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 Puppet master 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 will prevail.

Orchestrator configuration file settings

The orchestrator configuration file is formatted in JSON. For example:

```
{
    "options": {
        "service-url": "https://<PUPPET MASTER 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 Puppet master 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;PUPPET MASTER 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></td>
<td>• *nix: /etc/puppetlabs/puppet/ssl/certs/ca.pem</td>
</tr>
<tr>
<td></td>
<td>• Windows: C:\ProgramData\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 will use color. Set to true or false.</td>
</tr>
<tr>
<td>noop</td>
<td>Determines whether the orchestrator should run the Puppet agent in no-op mode. Set to true or false.</td>
</tr>
</tbody>
</table>

Disabling application management or orchestration services

Both application management and orchestration services are on by default in PE. If you need to disable these services, refer to Disabling application management and Disabling orchestration services.
Using Bolt

Bolt is an open source task runner that automates the manual work that you do to maintain your infrastructure.

Use Bolt to automate tasks that you perform on your infrastructure on an as-needed basis, for example, when you troubleshoot a system, deploy an application, or stop and restart services. Bolt connects directly to remote nodes with SSH or WinRM, so you are not required to install any agent software.

While you install Bolt separately from Puppet Enterprise, you can configure it to use orchestrator and run it on PE-managed nodes.

- **Using Bolt with orchestrator** on page 482
  Bolt enables running a series of tasks — called plans — to help you automate the manual work of maintaining your infrastructure. When you pair Bolt with PE, you get advanced automation with the management and logging capabilities of PE.

**Related information**

- Bolt Docs
- Puppet Tasks Hands-on Lab

Using Bolt with orchestrator

Bolt enables running a series of tasks — called plans — to help you automate the manual work of maintaining your infrastructure. When you pair Bolt with PE, you get advanced automation with the management and logging capabilities of PE.

Bolt connects directly to remote nodes with SSH or WinRM, so you are not required to install any agent software. To learn more about Bolt, see the Bolt documentation.

You can configure Bolt to use the orchestrator API and perform actions on PE nodes. When you run Bolt plans, the plan logic is processed locally on the node running Bolt while corresponding commands, scripts, tasks, and file uploads run remotely using the orchestrator API.

Before you can use Bolt with PE, you must install Bolt.

To set up Bolt to use the orchestrator API, you must:

- Install the bolt_shim module in a PE environment.
- Assign task permissions to a user role.
- Adjust the orchestrator configuration files, as needed.

Install the Bolt module in a PE environment

Bolt uses a task to execute commands, upload files, and run scripts over orchestrator. To install this task, install the puppetlabs-bolt_shim module from the Forge. Install the code in the same environment as the other tasks you want to run. Use the following Puppetfile line:

```puppet
mod 'puppetlabs-bolt_shim', '0.2.0'
```

In addition to the bolt_shim module, any task or module content you want to execute over Puppet Communications Protocol (PCP) must be present in the PE environment. For details about downloading and installing modules for Bolt, see Set up Bolt to download and install modules. By allowing only content that is present in the PE environment to be executed over PCP, you maintain role-based access control over the nodes you manage in PE.

To enable the Bolt apply action, you must install the puppetlabs-apply_helpers module. Use the following Puppetfile line:

```puppet
mod 'puppetlabs-apply_helpers', '0.1.0'
```

**Note:** Bolt over orchestrator can require a large amount of memory to convey large messages, such as the plugins and catalogs sent by apply. You might need to increase the Java heap size for orchestration services.
Assign task permissions to a user role

**CAUTION:** By granting users access to Bolt tasks, you give them permission to run arbitrary commands and upload files as a super-user.

1. In the console, click **Access control > User roles**.
2. From the list of user roles, click the role you want to have task permissions.
3. On the **Permissions** tab, in the **Type** box, select **Tasks**.
4. For **Permission**, select **Run tasks**, and select **All** from the **Instance** drop-down list.
5. Click **Add permission**, and commit the change.

Adjust the orchestrator configuration files

Set up the orchestrator API for Bolt in the same configuration file that is used for PE client tools:

- **nix** /etc/puppetlabs/client-tools/orchestrator.conf
- **Windows** C:/ProgramData/PuppetLabs/client-tools/orchestrator.conf

**Note:** If you use a global configuration file stored at /etc/puppetlabs/client-tools/orchestrator.conf (or C:\ProgramData\PuppetLabs\client-tools\orchestrator.conf for Windows), copy the file to your home directory.

Alternatively, you can configure Bolt to connect to orchestrator in the **pcp** section of the Bolt configuration file. This configuration is not shared with puppet task. By default, Bolt uses the production environment in PE when running tasks. To use a different environment, change the **task-environment** setting:

```yaml
pcp:
  task-environment: development
```

Specify the transport

Bolt runs tasks through the orchestrator when a target uses the **pcp** transport. Specify the transport for specific nodes by using the PCP protocol in the target's URI, like `pcp://puppet.certname`, or setting **transport** in a config section in `inventory.yaml`. Change the default transport for all nodes by setting **transport** in `bolt.yaml` or passing `--transport pcp` on the command line.

View available tasks

To view a list of available tasks from the orchestrator API, run the command `puppet task show` (instead of the command `bolt task show`).

Bolt Plan example

View a plan that combine multiple tasks with one command.

Plan that deploys an application

In this example, the plan `my_app` runs the tasks necessary to deploy an application to multiple nodes. It uses node information from an inventory file and tasks written in Python and stored at `my_app/tasks/`.

You run the plan with this command:

```
bolt plan run my_app::deploy version=1.0.2 app_servers=app db_server=db lb_server=lb --inventoryfile ./inventory.yaml --modulepath=./modules
```

Using the sample code below, the plan validates that there is a single load balancer server; queries the server load to determine availability, installs the application, migrates the database, makes the new code available on each application server and, finally, cleans up old versions of the application.
Note: This is sample code. To set up these tasks and run this plan in your environment, try the Puppet Tasks Hands-on Lab. This GitHub repository contains sample files, code examples, and exercises to help you interact with Bolt in a safe environment. For more information, see the puppetlabs/tasks-hands-on-lab repository.

```puppet
plan my_app::deploy(
  Pattern[/\d+\.\d+\.\d+/] $version,
  TargetSpec $app_servers,
  TargetSpec $db_server,
  TargetSpec $lb_server,
  String[1] $instance = 'my_app',
  Boolean $force = false
)
  # Validate that there is only a single load balancer server to check
  if get_targets($lb_server).length > 1 {
    fail_plan("{$lb_server} did not resolve to a single target")
  }

  # First query the load balancer and make sure the app isn't under too much
  # load to do a deploy.
  unless $force {
    $conns = run_task('my_app::lb', $lb_server,
      "Check load before starting deploy",
      action => 'stats',
      backend => $instance,
      server => 'FRONTEND',
    ).first['connections']
    if ($conns > 8) {
      fail_plan("The application has too many open connections: ${conns}")
    } else {
      # Info messages will be displayed when the --verbose flag is used.
      info("Application has ${conns} open connections.")
    }
  }

  # Install the new version of the application and check what version was
  # installed so it can be deleted after the deploy.
  $old_versions = run_task('my_app::install', [$app_servers, $db_server],
    "Install ${version} of the application",
    version => $version
  ).map |$r| { $r['previous_version'] }
  run_task('my_app::migrate', $db_server)

  # Don't log every action on each node, only log important messages
  without_default_logging() || {
    # Expand group references or globs before iterating
    get_targets($app_servers).each |$server| {
      # Check stats and print a message to the user
      $stats = run_task('my_app::lb', $lb_server,
        action => 'stats',
        backend => $instance,
        server => $server.name,
        _catch_errors => $force
      ).first
      notice("Deploying to ${server.name}, currently ${stats['status']} with
       ${stats['connections']} open connections.")

      run_task('my_app::lb', $lb_server,
        "Drain connections from ${server.name}",
        action => 'drain',
        backend => $instance,
      )
    }
  }
```

© 2020 Puppet, Inc.
server => $server.name,
      _catch_errors => $force
    )

run_task('my_app::deploy', [$server],
      "Update application for new version",
    )

# Verify the app server is healthy before returning it to the load
# balancer.
$health = run_task('my_app::health_check', $lb_server,
      "Run Healthcheck for ${server.name}",
      target => "http://${server.name}:5000/",
      _catch_errors' => true).first

if $health['status'] == 'success' {
  info("Upgrade Healthy, Returning ${server.name} to load balancer")
} else {
  # Fail the plan unless the app server is healthy or this is a forced
  deploy
  unless $force {
    fail_plan("Deploy failed on app server ${server.name}:
      ${health.result}"
      )
  }
}

run_task('my_app::lb', $lb_server,
      action => 'add',
      backend => $instance,
      server => $server.name,
      _catch_errors => $force
    )
notice("Deploy complete on ${server}.")
}

run_task('my_app::uninstall', [$db_server, $app_servers],
      "Clean up old versions",
      live_versions => $old_versions + $version,
    )
}

Related information
Puppet Tasks Hands-on Lab

Direct Puppet: a workflow for controlling change

The orchestrator—used alongside other PE tools, such as Code Manager—allows you to control when and how
infrastructure changes are made before they reach your production environment.

The Direct Puppet workflow gives you precise control over rolling out changes, from updating data and classifying
nodes, to deploying new Puppet code. In this workflow, you configure your agents to use cached catalogs during
scheduled runs, and you send new catalogs only when you're ready, via orchestrator jobs. Scheduled runs will
continue to enforce the desired state of the last orchestration job until you send another new catalog.

Related information
Enable cached catalogs for use with the orchestrator (optional) on page 479

© 2020 Puppet, Inc.
Enabling cached catalogs on your agents ensures Puppet will not enforce any catalog changes on your agents until you run an orchestrator job to enforce changes.

**Direct Puppet workflow**

In this workflow, you set up a node group for testing and validating code on a feature branch before you merge and promote it into your production environment.

**Before you begin**

- To use this workflow, you must enable cached catalogs for use with the orchestrator so that they enforce cached catalogs by default and compile new catalogs only when instructed to by orchestrator jobs.
- This workflow also assumes you’re familiar with Code Manager. It involves making changes to your control repo — adding or updating modules, editing manifests, or changing your Hiera data. You’ll also run deploy actions from the Code Manager command line tool and the orchestrator, so ensure you have access to a host with PE client tools installed.

**Related information**

[Installing PE client tools](#) on page 213

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily managed by Puppet.

**Set up node groups for testing new features**

The first step in the Direct Puppet workflow is to set up node groups for testing your new feature or code.

1. If they don't already exist, create environment node groups for branch testing, for example, you might 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: 1) the top-level parent environment node group, 2) node groups that represent your actual environments, and 3) node groups specific to feature testing.
3. In the **Rules** tab of the child node groups you created in the previous step, add this rule:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>agent_specified_environment</td>
</tr>
<tr>
<td>Operator</td>
<td>~</td>
</tr>
<tr>
<td>Value</td>
<td>^.+</td>
</tr>
</tbody>
</table>

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 366

Create custom environment node groups so that you can target deployment of Puppet code.

**Create a feature branch**

After you’ve set up a node group, create a new branch of your control repository on which you can make changes to your feature code.

1. Branch your control repository, and name the new branch, for example, `my_feature_branch`.

2. Make changes to the code on your feature branch. Commit and push the changes to the `my_feature_branch`.
Deploy code to the Puppet master and test it
Now that you've made some changes to the code on your feature branch, you're ready to use Code Manager to push those to the Puppet master.

1. To deploy the feature branch to the master, run the following Code Manager command: `puppet code deploy --wait my_feature_branch`

![Diagram of code deployment process]

**Note:** After this code deployment, there is a short delay while Puppet Server loads the new code.

2. To test your changes, run Puppet on a few agent-specified development nodes in the `my_feature_branch` environment, run the following orchestrator command:

   ```
   puppet job run --nodes my-dev-node1,my-dev-node2 --environment my_feature_branch
   ```

   **Tip:** You can also use the console to create a job targeted at a list of nodes in the `my_feature_branch` environment.

![Diagram of job execution process]

3. Validate your testing changes. Open the links in the orchestrator command output, or use the Job ID linked on the Job list page, to review the node run reports in the console. Ensure the changes have the effect you intend.

**Related information**
- Review jobs from the console on page 525
  View a list of recent jobs and drill down to see useful details about each one.
- Run Puppet on a node list on page 490
  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, and you're ready to promote your changes into production.

1. Merge `my_feature_branch` into the `production` branch in your control repo.
2. To deploy your updated production branch to the Puppet master, run the following Code Manager command:

```
puppet code deploy --wait production
```

**Preview the job**

Before running Puppet across the production environment, preview the job with the `puppet job plan` command.

To ensure the job will capture all the nodes in the production environment, as well as the agent-specified development nodes that just ran with the `my_feature_branch` environment, use the following query as the job target:

```
puppet job plan --query 'inventory {environment in ["production", "my_feature_branch"]}'
```

**Run the job on the production environment**

If you're satisfied with the changes in the preview, you're ready to enforce changes to the production environment.

Run the orchestrator job.

```
puppet job run --query 'inventory {environment in ["production", "my_feature_branch"]}'
```

**Tip:** You can also use the console to create a job targeted at this PQL query.

**Related information**

- [Run Puppet on a PQL query](#) on page 489
  For some jobs, you may want to target nodes that meet specific conditions. For such jobs, create a PQL query.

**Validate your production changes**

Finally, you're ready to validate your production changes.

Check the node run reports in the console to confirm that the changes were applied as intended. If so, you're done!

Repeat this process as you develop and promote your code.
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 in the console** on page 489
  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 493
  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 500
  Run the orchestrator across all nodes in an environment.

### Running Puppet on demand in 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 three ways to specify the job target (the nodes you want to run jobs on):

- A Puppet Query Language (PQL) query
- A static node list
- 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 PQL query

For some jobs, you may want to target nodes that meet specific conditions. For such jobs, create a PQL query.

Make sure you have permissions to run jobs and queries.

1. In the console, in the **Run** section, click **Puppet**.
2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.
3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes will 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, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

   **Note**: If your job target includes application instances, the selected environment will also determine the dependency order of your node runs.

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 run on all nodes in the job without enforcing a new catalog.
   - **Override noop = true configuration**: If any nodes in the job have `noop = true` set in their `puppet.conf` files, will ignore that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.
5. In the **Inventory** list, select **PQL query**.

6. 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](https://docs.puppet.com/puppet-query-language) 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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes</td>
<td>nodes[certname] { }</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Nodes with a specific fact and value (example: OS name is)</td>
<td>inventory[certname] { facts.os.name = &quot;&lt;OS&gt;&quot; }</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Nodes with a specific class (example: )</td>
<td>resources[certname] { type = &quot;Class&quot; and title = &quot;Apache&quot; }</td>
<td></td>
</tr>
<tr>
<td>Nodes assigned to a specific environment (example: production)</td>
<td>nodes[certname] { catalog_environment = &quot;production&quot; }</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Nodes with a specific resource and operating system (example: httpd and )</td>
<td>inventory[certname] { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }</td>
<td></td>
</tr>
</tbody>
</table>

7. Click **Submit query** and click **Refresh** to update the node results.

8. If you change or edit the query after it runs, click **Submit query** again.

9. Optional: To convert the 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.

10. Click **Run job**.

    **Important:** When you run this job, the PQL query will run again, and the job may 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.

### 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.

Make sure you have permissions to run jobs and queries.

1. In the console, in the **Run** section, click **Puppet**.
2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.
3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes will 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, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

   **Note**: If your job target includes application instances, the selected environment will also determine the dependency order of your node runs.

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 run on all nodes in the job without enforcing a new catalog.
   - **Override noop = true configuration**: If any nodes in the job have noop = true set in their puppet.conf files, will ignore that setting and enforce a new catalog on those nodes. This setting corresponds to the --no-noop flag available on the orchestrator CLI.

5. In the **Inventory** list, select **Node list**.

6. 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.

7. Select the nodes you want to add to the job. You can select nodes from multiple searches to create the node list target.

8. To remove any nodes from the target, select them and click **Remove selected**, or first click **Select all** and then click **Remove selected**.

9. Click **Run job**.

**Important**: When you run this job, the query will run again, and the job may 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.

### 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.

**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 will not appear if no rules have been specified for it.

1. In the console, in the **Run** section, click **Puppet**.

2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.

3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes will 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, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

   **Note**: If your job target includes application instances, the selected environment will also determine the dependency order of your node runs.

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 run on all nodes in the job without enforcing a new catalog.
   - **Override noop = true configuration**: If any nodes in the job have noop = true set in their puppet.conf files, will ignore that setting and enforce a new catalog on those nodes. This setting corresponds to the --no-noop flag available on the orchestrator CLI.
5. In the **Inventory** list, select **Node group**.
6. In the **Choose a node group** box, type or select a node group, and click **Select**.
7. Click **Run job**.

**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. It's likely that you'll 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. Once 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 <strong>Nodes with events</strong> 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 once.</td>
</tr>
</tbody>
</table>

Make sure you have permissions to run jobs and 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 job list target.
2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.
3. Select an environment:
   - **Run nodes in their own assigned environment**: Nodes will 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, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

   **Note**: If your job target includes application instances, the selected environment will also determine the dependency order of your node runs.

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 run on all nodes in the job without enforcing a new catalog.
   - **Override noop = true configuration**: If any nodes in the job have noop = true set in their puppet.conf files, will ignore 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 will clear the node list target.
6. Click **Run job**.
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 an on-demand Puppet run, runs that are underway will finish, and runs that have not started will not start.

Stopping tasks depends on how many nodes are targeted in the job versus the concurrency limit you’ve set. If the concurrency limit is higher than the number of nodes targeted, all nodes will complete the task, as those nodes will have already started the task.

To stop a job:
• In the console, go to the Job details page and select the Puppet run or Task tab. From the list of jobs, find the one you want and click Stop.
• On the command line, press CTRL + C.

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 tool 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.

If you run a job on a node that has relationships outside of the target (for example, it participates in an application that includes nodes not in the job target) the job will still only run on the node in the target you specified. In such cases, the orchestrator notifies you that external relationships exist. It prints the node with relationships, and it prints the applications that may be affected. For example:

**WARNING** target does not contain all nodes in this application.

You can run jobs on three types of targets, but these targets cannot be combined:
• An application or an application instance in an environment
• A list of nodes or a single node
• 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.

The orchestrator command line tool includes the puppet job and puppet app commands.
• puppet job
  The puppet job command is the main mechanism to control Puppet jobs. For a complete reference of the puppet job command, see:
  • Checking job plans on the CLI
  • Running jobs on the CLI
  • Reviewing jobs on the CLI
• puppet app
  The puppet app tool lets you view the application models and application instances you’ve written and stored on your Puppet master. For a complete reference of the puppet app command, see Reviewing applications on the CLI.
Run Puppet on 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
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 workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Make sure you have permissions to run jobs and queries.

Log into your Puppet master 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 queries you could run with the orchestrator.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single node by certname</td>
<td><code>--query 'nodes { certname = &quot;mynode&quot; }'</code></td>
</tr>
<tr>
<td>All nodes with &quot;web&quot; in certname</td>
<td><code>--query 'nodes { certname ~ &quot;web&quot; }'</code></td>
</tr>
<tr>
<td>All nodes</td>
<td><code>--query 'inventory { facts.os.name = &quot;CentOS&quot; }'</code></td>
</tr>
<tr>
<td>All nodes with httpd managed</td>
<td><code>--query 'inventory { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }'</code></td>
</tr>
<tr>
<td>All nodes with failed reports</td>
<td><code>--query 'reports { latest_report? = true and status = &quot;failed&quot; }'</code></td>
</tr>
<tr>
<td>All nodes matching the environment for the last received catalog</td>
<td><code>--query 'nodes { catalog_environment = &quot;production&quot; }'</code></td>
</tr>
</tbody>
</table>

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
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 workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Make sure you have permissions to run jobs and queries.

Log into your Puppet master 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.

**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 access to the nodes you want to target.

1. Log into your master 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 406

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

**Run Puppet on an application or an application instance in an environment**

Use applications as a job target to enforce Puppet runs in order on all nodes found in a specific application instance, or to enforce Puppet runs in order on all nodes that are found in each instance of an application.

**Before you begin**

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 workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Make sure you have permissions to run jobs and queries.

Log into your Puppet master or client tools workstation and run one of the following commands:

• To run a job on all instances of an application: `puppet job run --application <APPLICATION> --environment <ENVIRONMENT>`

• To run a job on an instance of an application in an environment: `puppet job run --application <APPLICATION INSTANCE> --environment <ENVIRONMENT>`

**Tip:** You can use `-a` in place of `--application`.

**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>--noop</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>Option</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--no-nopp</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 in puppet.conf</code>. Cannot be used in conjunction with <code>--noop</code> flag.</td>
</tr>
<tr>
<td>--environment, -e</td>
<td>Environment name</td>
<td>Overrides the environment specified in the orchestrator configuration file. The orchestrator uses this option to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Instruct nodes what environment to run in. If any nodes can’t run in the environment, those node runs will fail. A node will run in an environment if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The node is included in an application in that environment. These runs may fail if the node is classified into a different environment in the node classifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The node is classified into that environment in the node classifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Load the application code used to plan run order</td>
</tr>
<tr>
<td>--no-enforce-environment</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 puppet.conf files.</td>
</tr>
<tr>
<td>--description</td>
<td>Flag, defaults to empty</td>
<td>Provide a description for the job, which will be shown on the job list and job details pages, and returned with the puppet job show command.</td>
</tr>
<tr>
<td>--concurrency</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 will be 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, you can use the console to review a list of jobs or to view the details of jobs that have previously run or are in progress. Refer to Reviewing jobs in the PE console for more information.

**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 an on-demand Puppet run, runs that are underway will finish, and runs that have not started will not start.

Stopping tasks depends on how many nodes are targeted in the job versus the concurrency limit you've set. If the concurrency limit is higher than the number of nodes targeted, all nodes will complete the task, as those nodes will have already started the task.

To stop a job:

• In the console, go to the Job details page and select the Puppet run or Task tab. From the list of jobs, find the one you want and click Stop.
• On the command line, press CTRL + C.

**Viewing job plans**

The `puppet job plan` command allows you to preview a plan for a Puppet job without actually enforcing any change. Use this command to ensure your job will run as expected.

You can preview plans for three types of targets, but these targets cannot be combined:

• View a plan to enforce change on a list of nodes or a single node
• View a plan to enforce change based on a PQL nodes query
• View a plan to enforce change on an application or an application instance in an environment

**Results from the job plan command**

The `puppet job plan` command returns the following about a job:

• The environment the job will run in.
• The target for the job.
• The maximum concurrency setting for the job.
• The total number of nodes in the job run.
• A list of application instances included in the job, if applicable.
• A list of nodes sorted topographically, with components and application instances listed below each node. The node list is organized in levels by dependencies. Nodes shown at the top, level 0, have no dependencies. Nodes in level 1 have dependencies in level 0. A node can run once Puppet has finished running on all its dependencies.

**Note:** The `puppet job plan` does not generate a job ID. A job ID is only created, and shown in the job plan, when you use `puppet job run`. Use the job ID to view jobs with the `puppet job show` command.

After you view the plan:

• Use `puppet job run <SAME TARGET AND OPTIONS>` to create and run a job like this.
• Remember that node catalogs may have changed since this plan was generated.
• Review and address any errors that the service reports it will encounter when running the job. For example, it will detect any dependency cycles or components that not properly mapped to nodes.

© 2020 Puppet, Inc.
**Example job plan**

The following is an example plan summary for a job that will run on a list of nodes that contain a partial application instance.

```bash
$ puppet job plan --nodes db1.example.com, noapp.example.com, noapp2.example.com

<table>
<thead>
<tr>
<th>Environment</th>
<th>production</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>1</td>
</tr>
</tbody>
</table>

Application instances: 1
- lamp[example]

Node run order (nodes in each level can run simultaneously, nodes in level 0 will run before their dependent nodes in level 1, etc.):

```
0: db1.example.com
   lamp[example] = lamp[Db1[example]]
noapp.example.com
noapp2.example.com

Started puppet run on db1.example.domain.com ...
Started puppet run on web1.example.domain.com ...
Started puppet run on web2.example.domain.com ...
```

Use `puppet job run --nodes <NODE NAME>, <NODE NAME>, <NODE NAME>, <NODE NAME>` to create and run this job. Node catalogs may have changed since this plan was generated.

**View a job plan for a list of nodes or a single node**

You can view a plan for enforcing change on a single node or a comma-separated list of nodes.

**Before you begin**

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 workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Log into your Puppet master or client tools workstation and run one of the following commands:

- To view a plan for a single node: `puppet job plan --nodes <NODE NAME> <OPTIONS>`
- To view a plan for a list of nodes, use a **comma-separated** list of node names. `puppet job plan --nodes <NODE NAME>, <NODE NAME>, <NODE NAME>, <NODE NAME> <OPTIONS>`

**Tip:** You can use `-n` in place of `--nodes`.

**View a job plan for a PQL nodes query**

Use a PQL nodes query as a job target. In this case, the orchestrator presents a plan for a job that could run on a list of nodes returned from a PQL nodes query.

1. Ensure you have the correct permissions to use PQL queries.
2. Log into your Puppet master or client tools workstation and run the following command: `puppet job plan --query '<QUERY>' <OPTIONS>

Tip: You can use -q in place of --query.

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><code>--query 'nodes { certname = &quot;mynode&quot; }'</code></td>
</tr>
<tr>
<td>All nodes with &quot;web&quot; in certname</td>
<td><code>--query 'nodes { certname ~ &quot;web&quot; }'</code></td>
</tr>
<tr>
<td>All CentOS nodes</td>
<td><code>--query 'inventory { facts.os.name = &quot;CentOS&quot; }'</code></td>
</tr>
<tr>
<td>All CentOS nodes with httpd managed</td>
<td><code>--query 'inventory { facts.operatingsystem = &quot;CentOS&quot; and resources { type = &quot;Service&quot; and title = &quot;httpd&quot; } }'</code></td>
</tr>
<tr>
<td>All nodes with failed reports</td>
<td><code>--query 'reports { latest_report? = true and status = &quot;failed&quot; }'</code></td>
</tr>
<tr>
<td>All nodes matching the environment for the last received catalog</td>
<td><code>--query 'nodes { catalog_environment = &quot;production&quot; }'</code></td>
</tr>
</tbody>
</table>

**View a job plan for applications or application instances**

You can view a plan for enforcing change on all instances of an application or for a specific instance of an application.

Log into your Puppet master or client tools workstation and run one of the following commands:

- To view a job plan for all instances of an application: `puppet job plan --application <APPLICATION> <OPTIONS>`
- To view a job plan for an application instance in an environment: `puppet job plan --application <APPLICATION INSTANCE> <OPTIONS>`

Tip: You can use -a in place of --application.
Puppet job plan command options
The following table shows common options for the puppet job plan command. For a complete list of options, run puppet job --help.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| --environment, -e | Environment name | Overrides the environment specified in the orchestrator configuration file. The orchestrator uses this option to:  
  • Instruct nodes what environment to run in. If any nodes can’t run in the environment, those node runs will fail. A node will run in an environment if:  
    • The node is included in an application in that environment. These runs may fail if the node is classified into a different environment in the node classifier.  
    • The node is classified into that environment in the node classifier.  
    • Load the application code used to plan run order |
| --concurrency    | Integer    | Limits how many nodes can run concurrently. Default is unlimited. You can tune concurrent compile requests in the console. |

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. This key is required.</td>
</tr>
<tr>
<td>scope</td>
<td>Object, required unless target is specified. The PuppetDB query, a list of nodes, a classifier node group id, or an application/application instance 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>Integer, the maximum number of nodes to run at once. The default, if unspecified, is unlimited.</td>
</tr>
</tbody>
</table>
**Key** | **Definition**
--- | ---
enforce_environment | 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.

debug | Boolean, whether to use the `--debug` flag on Puppet agent runs.

trace | Boolean, whether to use the `--trace` flag on Puppet agent runs.

evaltrace | Boolean, whether to use the `--evaltrace` flag on Puppet agent runs.

target | String, required unless scope is specified. The name of an application or application instance to deploy. If an application is specified, all instances of that application will be deployed. If this key is left blank or unspecified without a scope, the entire environment will be deployed. 
**This key is deprecated.**

For example, to deploy the `node1.example.com` environment in no-op mode, the following request is valid:

```json
{
    "environment" : "",
    "noop" : true,
    "scope" : {
        "nodes" : ["node1.example.com"]
    }
}
```

**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 will be 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 <code>certname</code> 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 will be 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>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>-----</td>
<td>------------</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**

- Puppet orchestrator API: forming requests on page 529
  Instructions on interacting with this API.
- Puppet orchestrator API: error responses on page 562
  From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**Running tasks**

Use the orchestrator to set up jobs in the console or on the command line and run tasks across systems in your infrastructure.

You can install pre-existing tasks, and run tasks from the console and from the command line.

**Note:** If you have set up compile masters and you want to use tasks, you must either set master_urus or you server_list on agents to point to to your compile masters. This setting is described in the section on configuring compile masters for orchestrator scale.

- Running tasks in Puppet Enterprise on page 504
  Puppet Enterprise allows you to run arbitrary, or *ad hoc*, tasks (as opposed to systematic Puppet configuration management changes), and eliminate manual work across your infrastructure. A Puppet task is a single action that you execute on target machines by calling an executable file.
- Installing tasks on page 504
  Puppet Enterprise comes with some pre-installed tasks, but you must install any other tasks you want to use.
- Running tasks from the console on page 504
  PE gives you the ability to execute ad-hoc tasks on target machines from the console. Task are the quickest way 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 509
  Use the puppet task run command to run tasks on agent nodes.

**Related information**

- Configure compile masters on page 207

© 2020 Puppet, Inc.
Compile masters must be configured to appropriately route communication between your master of masters and agent nodes.

**Running tasks in Puppet Enterprise**

Puppet Enterprise allows you to run arbitrary, or *ad hoc*, tasks (as opposed to systematic Puppet configuration management changes), and eliminate manual work across your infrastructure. A Puppet task is a single action that you execute on target machines by calling an executable file.

With Puppet tasks, you can troubleshoot and deploy changes to individual or multiple systems in your infrastructure. Role-based access control (RBAC) determines who can run tasks on which parts of your infrastructure. You can delegate access, even on a per-task basis, to ensure that teams or individuals can run tasks only on infrastructure they manage. Additionally, built-in reporting shows a complete history of task jobs.

You can run tasks from your tool of choice: the console, the orchestrator command line interface (CLI), or the orchestrator API /command/task endpoint.

Whatever tool you choose to run tasks with, you'll have the ability to filter nodes to target portions of an environment for task runs based on any attributes or events, such as your Windows nodes in your European data center, or all nodes with a specific version of OpenSSL installed. You can launch a task and check back later on the success of the task with the console or CLI.

Tasks are a type of orchestrator job. You can run a task job from either the console or the orchestrator CLI, and you can run it against a list of nodes, or against a PQL nodes query.

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 will change 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 will be reset to whatever is specified in a manifest on the next Puppet run.

**Installing tasks**

Puppet Enterprise comes with some pre-installed tasks, but you must install any other tasks you want to use.

PE comes with `facter_task`, `package`, `service`, and `puppet_conf` tasks. Use the `package` task to inspect, install, upgrade, and manage packages. Use the `service` task to start, stop, restart, and check the status of services running on your systems.

Tasks are packaged in Puppet modules, so you can install them directly from the Forge or install and manage them with a Puppetfile and Code Manager.

If you plan to run tasks from the console, those tasks must be installed into the *production* environment.

**Related information**

*Managing environment content with a Puppetfile* on page 569

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 from the console**

PE gives you the ability to execute ad-hoc tasks on target machines from the console. Task are the quickest way 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 jobs on):

- A Puppet Query Language (PQL) query
- A static node list
- A node group

You can't combine these methods, and if you switch from one to the other with the *Inventory* 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.

**Important:** 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 (such as upgrading a service or package), a subsequent Puppet run will change 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 will be reset to whatever is specified in the relevant manifest on the next Puppet run.

**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 queries.

1. In the console, in the Run section, click **Task**.
2. 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.

3. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.

4. Set any parameters and values for the task. Click **Add parameter** for each parameter-value pair you add.

   **Note:** Parameters marked with an asterisk (*) are required. Optional parameters are available in a dropdown. If there are no required or optional parameters, you can add arbitrary parameters provided those parameters are allowed in the task.

   Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Wrap boolean values in double quotes (for example, "false").

   If you chose the **service** task, then you have two required parameters. For **action**, you can choose **restart**. For service, enter nginx.

5. Optional: To schedule the job to run at a future time, select **Later** and choose a start date and time.

6. In the **Inventory** list, 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.

<table>
<thead>
<tr>
<th>Target</th>
<th>PQL query</th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes</td>
<td>nodes<a href="%7B%7D">certname</a></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)</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:)</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 )</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 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. Run or schedule the job.
    - Run job
    - Schedule job

Important: When you run this job, the PQL query will run again, and the job may 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.

Run a task on a node list
Create a node list target 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 access to the nodes you want to target.
Make sure you have permissions to run tasks and queries.

1. In the console, in the **Run** section, click **Task**.

2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.

3. 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.

4. Set any parameters and values for the task. Click **Add parameter** for each parameter-value pair you add.
   
   **Note:** Parameters marked with an asterisk (*) are required. Optional parameters are available in a dropdown. If there are no required or optional parameters, you can add arbitrary parameters provided those parameters are allowed in the task.

   Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Wrap boolean values in double quotes (for example, "false").

   If you chose the service task, then you have two required parameters. For **action**, you can choose **restart**. For service, enter nginx.

5. Optional: To schedule the job to run at a future time, select **Later** and choose a start date and time.

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. To remove any nodes from the target, select them and click **Remove selected**, or first click **Select all** and then click **Remove selected**.

10. Run or schedule the job.

    - **Run job**
    - **Schedule job**

### 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 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 will not appear if no rules have been specified for it.

1. In the console, in the **Run** section, click **Task**.

2. 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.

3. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.
4. Set any parameters and values for the task. Click **Add parameter** for each parameter-value pair you add.

**Note:** Parameters marked with an asterisk (*) are required. Optional parameters are available in a dropdown. If there are no required or optional parameters, you can add arbitrary parameters provided those parameters are allowed in the task.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Wrap boolean values in double quotes (for example, "false").

If you chose the **service** task, then you have two required parameters. For **action**, you can choose **restart**.

For **service**, enter **nginx**.

5. Optional: To schedule the job to run at a future time, select **Later** and choose a start date and time.

6. In the **Inventory** list, select **Node group**.

7. In the **Choose a node group** box, type or select a node group, and click **Select**.

8. Run or schedule the job.

- **Run job**
- **Schedule job**

### Schedule a task

Schedule a job to run a task at a particular date and time.

**Before you begin**

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

1. In the console, in the **Run** section, click **Task**.

2. Optional: In the **Job description** field, provide a description that will be shown on the job list and job details pages.

3. 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.

4. Set any parameters and values for the task. Click **Add parameter** for each parameter-value pair you add.

**Note:** Parameters marked with an asterisk (*) are required. Optional parameters are available in a dropdown. If there are no required or optional parameters, you can add arbitrary parameters provided those parameters are allowed in the task.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Wrap boolean values in double quotes (for example, "false").

If you chose the **service** task, then you have two required parameters. For **action**, you can choose **restart**.

For **service**, enter **nginx**.

5. Under **Schedule**, select **Later** and choose a start date, time, and time zone for the job to run.

**Tip:** Based on the configuration of your console, the default time zone is either UTC, Coordinated Universal Time, or the local time, UTC offset, of your browser.

6. 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.

7. Click **Schedule job**.

Your task appears on the **Schedule** tab of the **Jobs** page.
Related information
Delete a scheduled job on page 509
Delete a job that is scheduled to run at a later time.
Review jobs from the console on page 525
View a list of recent jobs and drill down to see useful details about each one.

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 an on-demand Puppet run, runs that are underway will finish, and runs that have not started will not start.

Stopping tasks depends on how many nodes are targeted in the job versus the concurrency limit you've set. If the concurrency limit is higher than the number of nodes targeted, all nodes will complete the task, as those nodes will have already started the task.

To stop a job:
• In the console, go to the Job details page and select the Puppet run or Task 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, go to Jobs > Scheduled tab.
2. From the list of jobs, find the one you want to delete and click Remove.

Running tasks from the command line
Use the puppet task run command to run tasks on agent nodes.

Before you begin
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 workstation, you must specify the full path to the command. For example, c:\Program Files\Puppet Labs\Client\bin\puppet-task run.

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.

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 queries.

Log into your master or client tools workstation and run one of the following commands:
To specify the query on the command line: puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --query "<QUERY>" <OPTIONS>

To pass the query in a text file: 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:

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. Parameters passed inline as <PARAMETER>=<VALUE> will be treated as strings.

Refer to the puppet task command options to see how to pass parameters, particularly non-strings, with the --params flag.

The following table shows some example targets and the associated PQL queries you could run with the orchestrator.

<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 nodes</td>
<td>--query 'inventory {facts.os.name = &quot;CentOS&quot;}'</td>
</tr>
<tr>
<td>All 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>

Tip: You can use -q in place of --query.

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 queries.

Log into your master 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.
To run a task job on a node list from a text file:

```
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 nodes:

```
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. Parameters passed inline as `<PARAMETER>=<VALUE>` will be treated as strings.

Refer to the `puppet task` command options to see how to pass parameters, particularly non-strings, with the `--params` flag.

**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 master 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 406
- Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

**puppet task run command options**

The following are common options you might 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, @/path/to/file.json. 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>
<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>
</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

  **Important:** All parameters passed inline will be treated as strings. If you want, for example, `timeout=8` to be passed as an integer, use the `--params` option.

- With the `--params` option, as a JSON object or reference to a JSON file:

  puppet task run <TASK NAME> --params '<JSON OBJECT>' --nodes <LIST OF NODES>
puppet task run my_task --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 methods of passing in parameters; choose either inline or `--params`. If you use the inline method, all parameters are passed 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 will either be 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
```

© 2020 Puppet, Inc.
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.

Related information
Review jobs from the console on page 525
View a list of recent jobs and drill down to see useful details about each one.

Review jobs from the command line on page 527
Use the puppet job show command to view running or completed jobs.

Inspect tasks
View the tasks that you have installed and have permission to run, as well as the documentation for those tasks.

Before you begin
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 workstation, you must specify the full path to the command. For example, c:\Program Files\Puppet Labs\Client\bin\puppet-task run.

Install the tasks you want to use.

Make sure you have permissions to run tasks and queries.

Log into your master 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

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 an on-demand Puppet run, runs that are underway will finish, and runs that have not started will not start.

Stopping tasks depends on how many nodes are targeted in the job versus the concurrency limit you've set. If the concurrency limit is higher than the number of nodes targeted, all nodes will complete the task, as those nodes will have already started the task.

To stop a job:

- In the console, go to the Job details page and select the Puppet run or Task tab. From the list of jobs, find the one you want and click Stop.
- On the command line, press CTRL + C.
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.

```ruby
#!/opt/puppetlabs/puppet/bin/ruby
require 'json'
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 26

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.

© 2020 Puppet, Inc.
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

```plaintext
String $mode = 'file'
```

use

```plaintext
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.

Instead of

```plaintext
String $path
```

use

```plaintext
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

```plaintext
eval "echo $input"
```

use

```plaintext
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.

Task names must be unique. If there are two tasks with the same name but different file extensions in a module, the task runner won't load either of them.

Each task or plan name segment must begin with a lowercase letter and:

- 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.

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']}`
end
```

```json
myservice/tasks/start.json
{
  "description": "Start a service",
  "parameters": {
    "service": {
      "type": "String",
      "description": "The service to start"
    }
  },
  "implementations": [
    {"name": "init.rb"}
  ]
}
```

```json
myservice/tasks/stop.json
{
  "description": "Stop a service",
  "parameters": {
    "service": {
      "type": "String",
      "description": "The service to stop"
    }
  },
  "implementations": [
    {"name": "init.rb"}
  ]
}
```

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 should 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 519.

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.

```
#!/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.

```
[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 $result[0] -Compress
    } elseif ($result.Count -gt 1) {
        ConvertTo-Json -InputObject @{"_items" = $result} -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.

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.

```json
{ "_error": {  
    "msg": "Task exited 1: Something on stderr",  
    "kind": "puppetlabs.tasks/task-error",  
    "details": { "exitcode": 1 } }
}
```

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 may 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",  
    # namespace the error to this module  
    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.

© 2020 Puppet, Inc.
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 may 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=$1
if [ -z "$version" ]; then
  echo "Must specify a version to deploy" && exit 1
else
  filename=$2
```
filename=~/myfile
fi

To convert the script into a task, replace this logic with task variables:

```
version=$PT_version # no need to validate if we use metadata
if [ -z "$PT_filename" ]; then
    filename=$PT_filename
else
    filename=~/myfile
fi
```

### 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 noop mode.

### No-op metadata example

```
{
    "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

```
#!/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": {
```
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.

```json
{
  "description": "Allows you to execute arbitrary SQL",
  "input_method": "stdin",
  "parameters": {
    "database": {
      "description": "Database to connect to",
      "type": "Optional[String[1]]"
    },
    "user": {
      "description": "The user",
      "type": "Optional[String[1]]"
    },
    "password": {
      "description": "The password",
      "type": "Optional[String[1]]",
      "sensitive": true
    },
    "sql": {
      "description": "The SQL you want to execute",
      "type": "String[1]"
    }
  }
}
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:

```
"provider": {
  "description": "The provider to use to manage or inspect the service, defaults to the system service manager",
  "type": "Optional[String[1]]"
}
```

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.

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 2: 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>• environment • stdin • powershell</td>
<td>Both environment and stdin unless .ps1 tasks, in which case powershell</td>
</tr>
</tbody>
</table>
### Metadata key
- **“parameters”**: The parameters or input the task accepts listed with a puppet type string and optional description. See [adding parameters to metadata](#) for usage information.
- **“puppet_task_version”**: The version of the spec used.
- **“supports_noop”**: Whether the task supports no-op mode. Required for the task to accept the `--noop` option on the command line.

### 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 3: 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 <code>/\w+/</code> 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`:

   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 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 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:

```
{
   "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`

### Reviewing jobs

You can review jobs—on-demand Puppet run, task, plan, or scheduled—from the console or the command line.

- **Review jobs from the console** on page 525
  View a list of recent jobs and drill down to see useful details about each one.
- **Review jobs from the command line** on page 527
  Use the `puppet job show` command to view running or completed jobs.

#### Review jobs from the console

View a list of recent jobs and drill down to see useful details about each one.

In the console, click **Jobs**.

**Job list page**

The **Job list** page provides a quick glance at key information about jobs run from the console. It also gathers information for jobs run from the orchestrator command-line interface.

Jobs are organized by type: **Puppet run, Task, Plan** (a plan combines multiple tasks and runs them with a single Bolt command) and **Scheduled**. And they are sorted by the date and time of the job's most recent status change; when the job started, stopped, completed or is scheduled to run.

Puppet run, Task, and Plan jobs have one of the following statuses:

- In progress
- Succeeded
- Finished with failures
- Stopping
- Stopped

There’s also a link to the profile of the user who ran the job. If a user record has been deleted, the UUID for that user appears instead.

To examine a job's details, click its **Job ID**. Note that the **Job ID** is a link only if you have the correct permissions to run the related job.

© 2020 Puppet, Inc.
**Related information**

**User permissions** on page 277
The table below lists the available object types and their permissions, as well as an explanation of the permission and how to use it.

**Using Bolt with orchestrator** on page 482
Bolt enables running a series of tasks — called *plans* — to help you automate the manual work of maintaining your infrastructure. When you pair Bolt with PE, you get advanced automation with the management and logging capabilities of PE.

**Job details page**
The *Job details* page provides a variety of information about jobs run from the console or the orchestrator command line interface.

To open the details page for a job, click its *Job ID* on the Job list page. The Job details page captures the job start time, duration, and provides a link to the user who ran the job. The *Job Details* link shows the job’s inventory target, environment, concurrency (if set on a job run from the CLI), and run mode. For task jobs, the *Job Details* link show the job’s inventory target and any parameters and values set for the task.

There’s also a link to the profile of user who ran the job. If a user is deleted, the UUID for that user will show instead of their name.

On this page, jobs have one of the following statuses:

- Succeeded
- Failed
- Skipped

**Reviewing the node run results table**

In the *Node run results* table, node runs in the job are sorted by status and then by certname. The table shows the number of total resources affected on each node in the job, and also tracks the following event types for each node:

<table>
<thead>
<tr>
<th>Event type</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. To learn more about a failed node, click the node link and use the <em>Reports</em> page to drill down through its events.</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>Skipped resource</td>
<td>A prerequisite for this resource was not met. This prerequisite is either one of the resource’s dependencies or a timing limitation set with the schedule metaparameter. The resource might be in sync or out of sync; Puppet doesn’t know yet.</td>
</tr>
</tbody>
</table>

Additionally, you can use the *Job node status* filter to filter nodes in this table based on their status. For example, if a job has completed and there were failures, you can filter to view only failed nodes, so you know which nodes to investigate further.

Nodes can have the following statuses:

- Succeeded
- Failed
- Failed - error...
• Skipped
• In progress (Only used for running jobs.)
• Queued

To view a node's run report, click the link in its Report column.

To export job details to a CSV file, click Export data. The CSV includes the same information as the Node run results table.

**Reviewing the node run results table for tasks**

The node run results returned by the orchestrator will vary depending on the task type. If output is available for a task that completes successfully, it will include the number of nodes affected and any parameters you specified in the task. Output is shown in standard output (stdout) or structured output. For example, a exec task that runs the `ls` command on your nodes will show results in stdout output, but a package upgrade task will most likely show results in structured output.

Some tasks can successfully complete but have no output. In such cases, the orchestrator returns a `No output for this node` message.

When tasks fail, orchestrator will return a structured error that describes the failure.

**Rerunning jobs from the details page**

On the Job details page, you can run a job again, using the same settings specified in the original job. You must have the correct permissions to run Puppet or tasks to rerun a job.

To run a job again, click the Run again button. This action sets up a new job using the same settings from the original job. The new job recreates the PQL query or node list target and runs on the same nodes.

Jobs with application targets run from the orchestrator command line cannot be rerun from the console, as the console doesn't support these targets. Similarly, tasks run as part of a plan cannot be run from the console (a plan combines multiple tasks and runs them with a single Bolt command).

Additionally, the whole_environment target is deprecated, so jobs with that target cannot be rerun from the console or command line interface.

**Review jobs from the command line**

Use the `puppet job show` command to view running or completed jobs.

**Before you begin**

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 workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Additionally, you can view a list of jobs or view the details of jobs that have previously run or are in progress from the Job details page in the console.

1. Log into your Puppet master or client tools workstation.
2. Run one of the following commands:

- To check the status of a running or completed job, run: `puppet job show <job ID>`

  The command returns the following:
  - The status of the job (running, finished, or failed).
  - The job type: Puppet run, task, or plan task (a plan combines multiple tasks and runs them with a single Bolt command).
  - The job description, if set.
  - The start and finish time.
  - The elapsed time or duration of the run.
  - The user who ran the job.
  - The environment the job ran in, if you set the environment.
  - Whether no-op mode was enforced.
  - The number of nodes in the job.
  - The target specified for the job.

- To view a list of running and completed jobs, up to 50 maximum (the concurrency limit), ordered by timestamp, run: `puppet job show`

**Viewing jobs triggered without the puppet job command**

When you run any orchestrator jobs through the console or the orchestrator API, the `puppet job show` command also prints those jobs.

You can use the `puppet job show` command with or without a job ID.

```
puppet job show
```

<table>
<thead>
<tr>
<th>ID</th>
<th>STATUS</th>
<th>JOB TYPE</th>
<th>TIMESTAMP</th>
<th>TARGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>3435</td>
<td>failed</td>
<td>echo</td>
<td>04/04/2018 06:07:18 PM</td>
<td>192-168-2-171.rfc...</td>
</tr>
<tr>
<td>3434</td>
<td>finished</td>
<td>puppet run</td>
<td>04/04/2018 05:57:27 PM</td>
<td>911bcc40-4550-44e...</td>
</tr>
<tr>
<td>3433</td>
<td>finished</td>
<td>package</td>
<td>04/04/2018 05:53:14 PM</td>
<td>911bcc40-4550-44e...</td>
</tr>
<tr>
<td>3432</td>
<td>finished</td>
<td>echo</td>
<td>04/04/2018 05:38:36 PM</td>
<td>bronze-10,bronce-...</td>
</tr>
</tbody>
</table>

*environment not enforced on a per node basis

**The enforce_environment option**

The `enforce_environment` option is an option on the `/command/` API endpoint. By default `enforce_environment` is set to `true`, which means agent nodes are forced to run in the same environment in which their configured applications are defined. If set to false, agent nodes will run in whatever environment they are classified in or assigned to. API jobs may not have an environment specified. If a job has no environment specified, agent node run ordering is not specified by applications.

**Puppet orchestrator API v1 endpoints**

Use this API to gather details about the orchestrator jobs you run.
• Puppet orchestrator API: forming requests on page 529
  Instructions on interacting with this API.
• Puppet orchestrator API: command endpoint on page 530
  Use the /command endpoint to start and stop orchestrator jobs for tasks and plans.
• Puppet orchestrator API: events endpoint on page 538
  Use the /events endpoint to learn about events that occurred during an orchestrator job.
• Puppet orchestrator API: inventory endpoint on page 539
  Use the /inventory endpoint to discover which nodes can be reached by the orchestrator.
• Puppet orchestrator API: jobs endpoint on page 542
  Use the /jobs endpoint to examine orchestrator jobs and their details.
• Puppet orchestrator API: scheduled jobs endpoint on page 550
  Use the /scheduled_jobs endpoint to gather information about orchestrator jobs scheduled to run.
• Puppet orchestrator API: plan jobs endpoint on page 553
  Use the /plan_jobs endpoint to view details about plan jobs you have run.
• Puppet orchestrator API: tasks endpoint on page 558
  Use the /tasks endpoint to view details about the tasks pre-installed by PE and those you've installed.
• Puppet orchestrator API: root endpoint on page 561
  Use the /orchestrator endpoint to return metadata about the orchestrator API.
• Puppet orchestrator API: error responses on page 562
  From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: forming 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: Requests to the orchestrator API should be well-formed HTTP(S) requests.

Authenticating to the orchestrator API with a token
You need to authenticate requests to the orchestrators’s API. You can do this using user authentication tokens.

For detailed information about authentication tokens, see Token-based authentication on page 289. 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:

```
```

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:

```
```

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:


**Puppet orchestrator API: command endpoint**

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, a classifier node group id, or an application/application instance 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>Integer, the maximum number of nodes to run at once. The default, if unspecified, is unlimited.</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>target</td>
<td>String, required unless scope is specified. The name of an application or application instance to deploy. If an application is specified, all instances of that application will be deployed. If this key is left blank or unspecified without a scope, the entire environment will be deployed. <strong>This key is deprecated.</strong></td>
</tr>
</tbody>
</table>
For example, to deploy the `node1.example.com` environment in no-op mode, the following request is valid:

```json
{
    "environment": "",
    "noop": true,
    "scope": {
        "nodes": ["node1.example.com"]
    }
}
```

**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 will be 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 <code>certname</code> 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 will be 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**

Puppet orchestrator API: forming requests on page 529
Instructions on interacting with this API.

Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**POST /command/stop**

Stop an orchestrator job that is currently in progress. Puppet agent runs that are in progress will finish, but no new agent runs will start. While agents are finishing, the server will continue to produce events for the job.

The job will transition to status `stopped` once 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:

```json
{
    "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>
<tr>
<td>name</td>
<td>The name of the newly created job.</td>
</tr>
<tr>
<td>nodes</td>
<td>A hash that shows all of the possible node statuses, and how many nodes are currently in that status.</td>
</tr>
</tbody>
</table>

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>Key</th>
<th>Definition</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**

**Puppet orchestrator API: error responses** on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**POST /command/task**

Run a permitted task job across a set of nodes.

**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>environment</td>
<td>The environment to load the task from. The default is production.</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. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>description</td>
<td>A description of the job.</td>
</tr>
<tr>
<td>noop</td>
<td>Whether to run the job in no-op mode. The default is false.</td>
</tr>
<tr>
<td>task</td>
<td>The task to run on the targets. <strong>This key is required.</strong></td>
</tr>
<tr>
<td>params</td>
<td>The parameters to pass to the task. This key is required, but can be an empty object.</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>nodes</td>
<td>An array of node names to target.</td>
</tr>
<tr>
<td>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>query</td>
<td>A PuppetDB or PQL query to use to discover nodes. The target is built from the <code>certname</code> 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"
  }
}
```

**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>
<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/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>
<tr>
<td>puppetlabs.orchestrator/not-permitted</td>
<td>This error occurs when a user does not have permission to run the task on the requested nodes. Server returns a 403 response.</td>
</tr>
</tbody>
</table>

### Related information

**Puppet orchestrator API: error responses** on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

### 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 will be thrown. Required.</td>
</tr>
<tr>
<td>description</td>
<td>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>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"
}
```

### Response format

If the task schedules successfully the server returns 202.

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
{
    "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 scheduled_time provided is in the past, the server returns a 400 response.</td>
</tr>
</tbody>
</table>

### Related information

[Puppet orchestrator API: error responses](#) on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**Puppet orchestrator API: events endpoint**

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 ( n )th 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. See event-types.</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",
        "details" : {
            "node" : "puppet-agent.example.com",
            "detail" : {
                "noop" : false
            }
        },
        "message" : "Started puppet run on puppet-agent.example.com ..."
    }]
}
```
Event types

The response format for each event will contain 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>
</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

Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: inventory endpoint

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" : [
    {
```

© 2020 Puppet, Inc.
"name" : "foo.example.com",
"connected" : true,
"broker" : "pcp://broker1.example.com/server",
"timestamp": "2016-010-22T13:36:41.449Z"
},
{
"name" : "bar.example.com",
"connected" : true,
"broker" : "pcp://broker2.example.com/server",
}
]

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
Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll 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 <code>true</code> or <code>false</code>.</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:

```
{
    "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
Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**POST /inventory**  
Check if the given list of nodes is connected to the PCP broker.

**Request format**  
The request body is a JSON object specifying a list of nodes to check. For example:

```json
{
    "nodes" : [
        "foo.example.com",
        "bar.example.com",
        "baz.example.com"
    ]
}
```

**Response format**  
The response is a JSON object with a record for each node in the request, 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://broker.example.com/server",
            "timestamp" : "2017-07-14T15:57:33.640Z"
        },
        {
            "name" : "bar.example.com",
            "connected" : false
        },
        {
            "name" : "baz.example.com",
            "connected" : true,
            "broker" : "pcp://broker.example.com/server",
            "timestamp" : "2017-07-14T15:41:19.242Z"
        }
    ]
}
```

**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
Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: jobs endpoint
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 ( \text{owner}, \text{timestamp}, \text{environment}, \text{name}, ) and ( \text{state} ). Orderings requested on ( \text{owner} ) will be applied to the ( \text{login} ) subfield of ( \text{owner} ).</td>
</tr>
<tr>
<td>order</td>
<td>Indicates whether results should be returned in ascending or descending order. Valid values are &quot;asc&quot; and &quot;desc&quot;. Defaults to &quot;asc&quot;</td>
</tr>
</tbody>
</table>

For example:


Response format
The response is a JSON object that lists orchestrator jobs and associated details, and uses these 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 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: ( \text{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>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>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| description  | (deprecated) A user-provided description of the job. For future compatibility
                  the description in options should be used.                        |
| timestamp    | The time when the job state last changed.                                  |
| environment  | (deprecated) The environment that the job operates in.                      |
| node_count   | The number of nodes the job will run on.                                   |
| node_states  | A JSON map containing the counts of nodes involved with the job by current
                  node state. Unrepresented states will not be displayed. This field
                  will be null when no nodes exist for a job.                         |
| nodes        | A link to get more information about the nodes participating in a given job.|
| report       | A link to the report for a given job.                                      |
| events       | A link to the events for a given job.                                      |

For example:

```json
{
  "items" : [
    {
      "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
      "name": "1234",
      "state" : "finished",
      "command" : "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" : {
        "id" : "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
        "login" : "brian"
      },
      "description" : "deploy the web app",
      "timestamp": "2016-05-20T16:45:31Z",
      "environment" : { "name" : "production" },
      "events" : {
```
"id" : "https://localhost:8143/orchestrator/v1/jobs/375/events"
 },
 "nodes" : {
   "id" : "https://localhost:8143/orchestrator/v1/jobs/375/nodes"
 }
 },
...

"pagination": {
   "limit": 20,
   "offset": 0,
   "total": 42
 }
)
...

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
Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll 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>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 will run 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>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------</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>
</tbody>
</table>

For example:

```json
{
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
    "name": "1234",
    "command": "deploy",
    "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": {
        "id": "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
        "login": "admin"
    },
    "description": "deploy the web app",
    "timestamp": "2016-05-20T16:45:31Z",
    "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" },
    "report": { "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234/report" }
}
```

**Error responses**

For this endpoint, the kind key of the error displays the conflict.
### Key Definition

<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

**Puppet orchestrator API: error responses** on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**GET /jobs/:job-id/nodes**
List all of the nodes associated with a given job.

### Response format

The response is a JSON object that details the nodes associated with a given 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 <code>start_timestamp</code> and <code>finish_timestamp</code>.</td>
</tr>
<tr>
<td>start_timestamp</td>
<td>The time the node starting running or <code>nil</code> 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 <code>nil</code> 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 since the node started running if it is currently running, or <code>nil</code> if the node hasn't started running or was skipped.</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 nodes 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 will vary.</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",
            "start_timestamp": "2015-07-13T20:36:13Z",
            "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 should contain metrics from the puppet run.

For example:

An error when running a task:

```
"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"
}

Details

The details field contains information based on the last event and current state of the node and might be empty. In some cases it might duplicate data from the results key for historical reasons.

If the node state is finished or failed the details hash might 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!"
    }
  }},
  "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.

```json
{
   "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).

```json
{
   "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**

Puppet orchestrator API: error responses on page 562
From time to time, you might encounter an error using the orchestrator API. In such cases, you'll 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": "wss6c3w9wngpymc.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.

<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>hpuppetlabs.orchestrator/unknown-job</td>
<td>If the job does not exist, the server returns a 404 response.</td>
</tr>
</tbody>
</table>

**Related information**

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

**Puppet orchestrator API: scheduled jobs endpoint**

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>
</tbody>
</table>

For example:

```
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 task)</td>
</tr>
<tr>
<td>task</td>
<td>The name of the task associated with the scheduled job</td>
</tr>
<tr>
<td>scope</td>
<td>The specification of the targets for the task.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment that the job operates in.</td>
</tr>
<tr>
<td>owner</td>
<td>The specification for the user that requested the job.</td>
</tr>
<tr>
<td>description</td>
<td>A user-provided description of the job.</td>
</tr>
<tr>
<td>scheduled_time</td>
<td>An ISO8601 specification of when the scheduled job will run</td>
</tr>
<tr>
<td>noop</td>
<td>True if the job should run in no-operation mode, false otherwise</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>A restricted number of items for the request to return</td>
</tr>
<tr>
<td>offset</td>
<td>A number 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": [
      {
```

© 2020 Puppet, Inc.
"id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/10",
"name": "10",
"type": "task",
"scope": {"nodes": ["foo.example.com", "bar.example.com"]},
"environment": "production",
"owner": {
   "id": "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
   "login": "fred"
},
"description": "front face the nebaclouser",
"scheduled_time": "2027-05-05T19:50:08.000Z",
"noop": false,

"id": "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/9",
"name": "9",
"type": "task",
"scope": {"nodes": ["east.example.com", "west.example.com"]},
"environment": "production",
"owner": {
   "id": "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
   "login": "fred"
},
"description": "rear face the cranfitouser",
"scheduled_time": "2027-05-05T19:55:08.000Z",
"noop": false,

"pagination": {
   "limit": 2,
   "offset": 5,
   "total": 15
}

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

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

DELETE /scheduled_jobs/:job-id

Delete a scheduled job.

- Response 204
- Response 403
  - Body

   
   
   "kind": "puppetlabs.orchestrator/not-permitted",
   "msg": "Not authorized to delete job {id}"

© 2020 Puppet, Inc.
Puppet orchestrator API: plan jobs endpoint

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>
</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>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>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>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>Key</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------</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" : null,
   "name" : "37",
   "events" : {
      "id" : "https://localhost:50310/orchestrator/v1/plan_jobs/37/events"
   },
   "state" : "running",
   "id" : "https://localhost:50310/orchestrator/v1/plan_jobs/37",
   "created_timestamp" : "YYYY-MM-DDT20:22:08Z",
   "options" : {
      "description" : "Testing myplan",
      "plan_name" : "myplan",
      "parameters" : {
         "nodes" : [ "localhost" ]
      }
   },
   "owner" : {
      "email" : "",
      "is_revoked" : false,
      "last_login" : "YYYY-MM-DDT20:22:06.327Z",
      "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
   },
   "finished_timestamp" : null,
   "name" : "36",
   "events" : {
      "id" : "https://localhost:50310/orchestrator/v1/plan_jobs/36/events"
   },
   "state" : "running",
   "id" : "https://localhost:50310/orchestrator/v1/plan_jobs/36",
   "created_timestamp" : "YYYY-MM-DDT20:22:08Z",
   "options" : {
      "description" : "Testing myplan",
      "plan_name" : "myplan",
      "parameters" : {
         "nodes" : [ "localhost" ]
      }
   },
   "owner" : {
      "email" : "",
      "is_revoked" : false,
      "last_login" : "YYYY-MM-DDT20:22:06.327Z",
      "is_remote" : false,
      "login" : "admin",
      "is_superuser" : true,
      "id" : "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
      "role_ids" : [ 1 ],
      "display_name" : "Administrator",
      "is_group" : false
   }
},
```

© 2020 Puppet, Inc.
"is_group": false,
"result": null
},
{
"finished_timestamp": null,
"name": "35",
"events": {
   "id": "https://localhost:50310/orchestrator/v1/plan_jobs/35/events"
},
"state": "running",
"id": "https://localhost:50310/orchestrator/v1/plan_jobs/35",
"created_timestamp": "YYYY-MM-DDT20:22:07Z",
"options": {
   "description": "Testing myplan",
   "plan_name": "myplan",
   "parameters": {
      "nodes": [ "localhost" ]
   }
},
"owner": {
   "email": "",
   "is_revoked": false,
   "last_login": "YYYY-MM-DDT20:22:06.327Z",
   "is_remote": false,
   "login": "admin",
   "is_superuser": true,
   "id": "42bf351c-f9ec-40af-84ad-e976fcd7f4bd",
   "role_ids": [ 1 ],
   "display_name": "Administrator",
   "is_group": false
},
"result": null
},
{
"finished_timestamp": null,
"name": "34",
"events": {
   "id": "https://localhost:50310/orchestrator/v1/plan_jobs/34/events"
},
"state": "running",
"id": "https://localhost:50310/orchestrator/v1/plan_jobs/34",
"created_timestamp": "YYYY-MM-DDT20:22:07Z",
"options": {
   "description": "Testing myplan",
   "plan_name": "myplan",
   "parameters": {
      "nodes": [ "localhost" ]
   }
},
"owner": {
   "email": "",
   "is_revoked": false,
   "last_login": "YYYY-MM-DDT20:22:06.327Z",
   "is_remote": false,
   "login": "admin",
   "is_superuser": true,
   "id": "42bf351c-f9ec-40af-84ad-e976fcd7f4bd",
   "role_ids": [ 1 ],
   "display_name": "Administrator",
   "is_group": false
},
"result": null
},
{
"finished_timestamp": null,
"name": "33",
"events": {
   "id": "https://localhost:50310/orchestrator/v1/plan_jobs/33/events"
},
"state": "running",
"id": "https://localhost:50310/orchestrator/v1/plan_jobs/33",
"created_timestamp": "YYYY-MM-DDT20:22:07Z",
"options": {
   "description": "Testing myplan",
   "plan_name": "myplan",
   "parameters": {
      "nodes": [ "localhost" ]
   }
},
"owner": {
   "email": "",
   "is_revoked": false,
   "last_login": "YYYY-MM-DDT20:22:06.327Z",
   "is_remote": false,
   "login": "admin",
   "is_superuser": true,
   "id": "42bf351c-f9ec-40af-84ad-e976fcd7f4bd",
   "role_ids": [ 1 ],
   "display_name": "Administrator",
   "is_group": false
},
"result": null
},
{
"finished_timestamp": null,
"name": "33",
"events": {
   "id": "https://localhost:50310/orchestrator/v1/plan_jobs/33/events"
},
"state": "running",
"id": "https://localhost:50310/orchestrator/v1/plan_jobs/33",
"created_timestamp": "YYYY-MM-DDT20:22:07Z",
"options": {
   "description": "Testing myplan",
   "plan_name": "myplan",
   "parameters": {
      "nodes": [ "localhost" ]
   }
},
"owner": {
   "email": "",
   "is_revoked": false,
"events" : {  
   "id" : "https://localhost:50310/orchestrator/v1/plan_jobs/33/events"  
},  
"state" : "running",  
"id" : "https://localhost:50310/orchestrator/v1/plan_jobs/33",  
"created_timestamp" : "YYYY-MM-DDT20:22:07Z",  
"options" : {  
   "description" : "Testing myplan",  
   "plan_name" : "myplan",  
   "parameters" : {  
      "nodes" : [ "localhost" ]  
   }  
},  
"owner" : {  
   "email" : "",  
   "is_revoked" : false,  
   "last_login" : "YYYY-MM-DDT20:22:06.327Z",  
   "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  
},  
"pagination" : {  
   "limit" : 5,  
   "offset" : 3,  
   "total" : 40  
}  
}  

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>
</tbody>
</table>
### Key | Definition
--- | ---
timestamp | The time when the plan job state last changed.
created_timestamp | The time the plan job was created.
finished_timestamp | The time the plan job finished.
events | A link to the events for a given plan job.
status | A hash of jobs that ran as part of the plan job, with associated lists of states and their enter and exit times.

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",
        "is_group": false
    },
    "timestamp": "YYYY-MM-DDT16:45:31Z",
    "status": {
        "1": [
            {
                "state": "running",
                "enter_time": "YYYY-MM-DDT18:44:31Z",
                "exit_time": "YYYY-MM-DDT18:45:31Z"
            },
            {
                "state": "finished",
                "enter_time": "YYYY-MM-DDT18:45:31Z",
                "exit_time": null
            }
        ],
        "2": [
            {
                "state": "running",
                "enter_time": "YYYY-MM-DDT18:44:31Z",
                "exit_time": "YYYY-MM-DDT18:45:31Z"
            }
        ]
    }
}
```
Error responses

For this endpoint, the kind key of the error displays the conflict.

- puppetlabs.orchestrator/validation-error: if the job-id in the request is not an integer, the server returns a 400 response.
- puppetlabs.orchestrator/unknown-job: if the plan job does not exist, the server returns a 404 response.

Related information

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: tasks endpoint

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",
```
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>

Related information

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll 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 should be 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>
</tbody>
</table>
### Key

<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>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 should determine 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",
    "code_id": "urn:puppet:code-id:1:a86da166c30f871823f9b2ea224796e834840676;production"
  },
  "metadata": {
    "description": "Install a package",
    "supports_noop": true,
    "input_method": "stdin",
    "parameters": {
      "name": {
        "description": "The package to install",
        "type": "String[1]"
      },
      "provider": {
        "description": "The provider to use to install the package",
        "type": "Optional[String[1]]"
      },
      "version": {
        "description": "The version of the package to install, defaults to latest",
        "type": "Optional[String[1]]"
      }
    }
  },
  "files": [
    {
      "filename": "install",
      "uri": {
        "path": "/package/tasks/install",
        "params": {
          "environment": "production"
        }
      },
      "sha256": "a9089b5b9720dca38a49db6f164cf8a053a7ea528711325da1c23de94672980f"
    }
  ]
}
```

© 2020 Puppet, Inc.
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

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: root endpoint

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:

```
{
  "info": {
    "title": "Application Management API (EXPERIMENTAL)",
    "description": "Multi-purpose API for performing application management operations",
    "warning": "This version of the API is experimental, and may 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"
    }
  ]
}```
Error responses

For this endpoint, the server returns a 500 response.

Related information

Puppet orchestrator API: error responses on page 562

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll receive an error response.

Puppet orchestrator API: error responses

From time to time, you might encounter an error using the orchestrator API. In such cases, you'll 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"
  }
}
```

Managing and deploying Puppet code

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

Code management tools use Git version control to track, maintain, and deploy your Puppet modules, manifests, and data. This allows you to more easily maintain, update, review, and deploy Puppet code and data for multiple environments.
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 then Code Manager creates environments, installs modules, and deploys the new code to your masters, without interrupting agent runs.

If you are already using r10k to manage your Puppet code, we suggest that you upgrade to Code Manager. Code Manager works in concert with r10k, so when you switch to Code Manager, you no longer interact directly with r10k.

If you're using r10k and aren't ready to switch to Code Manager yet, you can continue using r10k alone. You push your code changes to your version control repo, deploy environments from the command line, and r10k creates environments and installs the modules for each one.

Both Code Manager and the r10k code management tool are built into PE, so you don't have to install anything. To begin managing your code with either of these tools, you'll perform the following tasks:

1. Create a control repository with Git for maintaining your environments and code.
   
   The control repository is the Git repository that code management uses to maintain and deploy your Puppet code and data and to create environments in your Puppet infrastructure. As you update your control repo, code management keeps each of your environments updated.

2. Set up a Puppetfile to manage content in your environment.
   
   This file specifies which modules and data to install in your environment, including what version of that content should be installed, and where to download the content from.

3. Configure Code Manager (recommended) or r10k.
   
   Configure code management in the console's master profile. If you need to customize your configuration further, you can do so by adding keys to Hiera.

4. Deploy environments. With Code Manager, either set up a deployment trigger (recommended), or plan to trigger your deployment from the command line, which is useful for initial configuration. If you are using r10k alone, you'll run it from the command line whenever you want to deploy.

The following sections go into detail about setting up and using Code Manager and r10k.

- Managing environments with a control repository on page 563
  
  To manage your Puppet code and data with either 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 569
  
  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 574
  
  Code Manager automates the management and deployment of your Puppet code. Push code updates to your source control repo, and then Puppet syncs the code to your masters, so that all your servers start running the new code at the same time, without interrupting agent runs.

- Managing code with r10k on page 608
  
  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.

- About file sync on page 620
  
  File sync helps Code Manager keep your Puppet code synchronized across multiple masters.

Managing environments with a control repository

To manage your Puppet code and data with either 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 repo) is the Git repository that code management uses to manage environments in your infrastructure. As you update code and data in your control repo, code management keeps each of your environments updated.
Code management creates and maintains your 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 with its own version of your Puppet code and data.

Environments are created in `/etc/puppetlabs/code/environments` on the master. To learn more about environments in Puppet, read the documentation about environments.

To create a control repo that includes the standard recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations. If you cannot access the internet or cannot use modules directly from the Forge because of your organization's security rules, create an empty control repo and add the files you need to it.

**Note:** For Windows systems, be sure your version control is configured to use CRLF line endings. See your version control system for instructions on how to do this.

At minimum, a control repo comprises:

- A Git remote repository. The remote is where your control repo is stored on your Git host.
- A default branch named production, rather than the usual Git default of master.
- A Puppetfile to manage your environment content.
- An `environment.conf` file that modifies the `$modulepath` setting to allow environment-specific modules and settings.

**CAUTION:** Enabling code management means that Puppet manages the environment directories and **existing environments are not preserved**. Environments with the same name as the new one are overwritten. Environments not represented in the control repo are erased. If you were using environments before, commit any necessary files or code to the appropriate new control repo branch, or back them up somewhere **before** you start configuring code management.

### Create a control repo from the Puppet template

To create a control repo that includes a standard recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

To base your control repo on the Puppet control repository template, you'll 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.

The control repo template contains the files needed to get you started with a functioning control repo, including:

- An `environment.conf` file to implement a `site-modules/` directory for roles, profiles, and custom modules.
- `config_version` scripts to notify you which control repo version was applied to the agents.
- Basic code examples for setting up roles and profiles.
- An example `hieradata` directory that matches the default hierarchy in PE.
1. Generate a private SSH key to allow access to the control repository.

   This SSH key cannot require a password.

   The pe-puppet user, which is created during installation, must be able to access and use the key file. If you are preparing your control repo before installing, place the key in the SSH key directory after installation.

   a) Make a directory for the SSH keys.

      ```
      mkdir /etc/puppetlabs/puppetserver/ssh
      ```

   b) Generate the key pair.

      ```
      ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
      ```

   c) Ensure that the pe-puppet user owns the SSH key directory.

      ```
      chown pe-puppet:pe-puppet /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
      ```

   d) Ensure that the pe-puppet user has read, write, and execute permissions for the files in the SSH key directory.

      ```
      chmod 755 /etc/puppetlabs/puppetserver/ssh/
      ```

   Your keys are now located in /etc/puppetlabs/puppetserver/ssh:

   • Private key: /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
   • Public key: /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub

   Configure your host to use the SSH public key you generated. The process to do this is different for every host. Usually, you create a user or service account, and then assign the SSH public key to it.

   Code management needs read access to your control repository, as well as any module repositories referenced in the.

   See the your host docs for detailed instructions on adding SSH keys to your server. Commonly used hosts include:

   • GitHub
   • BitBucket Server
   • GitLab

2. Create a repository in your account, with the name you want your control repo to have.

   Steps for creating a repository vary, depending on what host you are using (, GitLab, Bitbucket, or another provider). See your host’s documentation for complete instructions.

   For example, on:

   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.

3. From the command line, clone the Puppet control-repo template.

   ```
   git clone https://github.com/puppetlabs/control-repo.git
   ```

4. Remove the template repository as your default source.

   ```
   git remote remove origin
   ```
5. Add the control repository you created as the default source.

    git remote add origin <URL OF YOUR GIT REPOSITORY>

You now have a control repository based on the Puppet control-repo template. When you make changes to this repo on your workstation, push those changes to the remote copy of the control repo on your Git server, so that code management can deploy your infrastructure changes.

After you’ve set up your control repo, create a for managing your environment content with code management.

If you already have a, you can now configure code management. Code management configuration steps differ, depending on whether you’re using (recommended) or . For important information about the function and limitations of each of these tools, along with configuration instructions, see the and pages.

**Note:** For Bitbucket, you must preface your remote URL with `ssh://`, to match the format of Bitbucket’s SSH URLs. For example, `ssh://git@ghost.domain.com/pup/puppet.git`.

### Create an empty control repo

If you can’t use the control repo template because, for example, you cannot access the internet or use modules directly from the Forge because of your security rules, create an empty control repo and then add the files you need.

To start with an empty control repo, you create a new repo on your Git host and then copy it to your workstation. You make some changes to your repo, including adding a configuration file that allows code management tools to find modules in both your site and environment-specific module directories. When you’re done making changes, push your changes to your repository on your Git host.
1. Generate a private SSH key to allow access to the control repository.
   This SSH key cannot require a password.

   The `pe-puppet` user, which is created during installation, must be able to access and use the key file. If you are preparing your control repo before installing, place the key in the SSH key directory after installation.

   a) Make a directory for the SSH keys.

   ```bash
   mkdir /etc/puppetlabs/puppetserver/ssh
   ```

   b) Generate the key pair.

   ```bash
   ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
   ```

   c) Ensure that the `pe-puppet` user owns the SSH key directory.

   ```bash
   chown pe-puppet:pe-puppet /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
   ```

   d) Ensure that the `pe-puppet` user has read, write, and execute permissions for the files in the SSH key directory.

   ```bash
   chmod 755 /etc/puppetlabs/puppetserver/ssh/
   ```

   Your keys are now located in `/etc/puppetlabs/puppetserver/ssh`:
   - Private key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
   - Public key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub`

   Configure your host to use the SSH public key you generated. The process to do this is different for every host. Usually, you create a user or service account, and then assign the SSH public key to it.

   Code management needs read access to your control repository, as well as any module repositories referenced in the .

   See the your host docs for detailed instructions on adding SSH keys to your server. Commonly used hosts include:
   - GitHub
   - BitBucket Server
   - GitLab

2. Create a repository in your account, with the name you want your control repo to have.

   Steps for creating a repository vary, depending on what host you are using (GitHub, GitLab, Bitbucket, or another provider). See your host's documentation for complete instructions.

   For example, 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.

3. Clone the empty repository to your workstation by running `git clone <REPOSITORY URL>`

4. Create a file named `environment.conf` in the main directory of your control repo.

5. In your text editor, open `environment.conf` file, and add the line below to set the modulepath. Save and close the file.

   ```
   modulepath=site-modules:modules:$basemodulepath
   ```
6. Add and commit your change to the repository by running:
   a) `git add environment.conf`
   b) `git commit -m "add environment.conf"

   The `environment.conf` file allows code management tools to find modules in both your site and
   environment-specific module directories. For details about this file, see the `environment.conf` documentation.

7. Rename the master branch to production by running `git branch -m master production`

   **Important:** The default branch of a control repo must be production.

8. Push your repository's production branch from your workstation to your host by running `git push -u origin production`

   When you make changes to this repo on your workstation, push those changes to the remote copy of the control repo
   on your Git server, so that code management can deploy your infrastructure changes.

   After you've set up your control repo, create a for managing your environment content with code management.

   If you already have a, you can now configure code management. Code management configuration steps differ,
   depending on whether you're using (recommended) or . For important information about the function and limitations
   of each of these tools, along with configuration instructions, see the and pages.

   **Note:** For Bitbucket, you must preface your remote URL with `ssh://`, to match the format of Bitbucket’s SSH
   URLs. For example, `ssh://git@github.com/pup/puppet.git`.

**Add an environment**

Create new environments by creating branches based on the production branch of your control repository.

**Before you begin**

Make sure you have:

- Configured either Code Manager or r10k.
- Created a Puppetfile in the default (usually 'production') branch of your control repo.
- Selected your code management deployment method (such as the `puppet-code` command or a webhook).

**Remember:** If you are using multiple control repos, do not duplicate branch names unless you use a source prefix.
For more information about source prefixes, see the documentation about configuring sources.

1. Create a new branch: `git branch <NAME-OF-NEW-BRANCH>`
2. Check out the new branch: `git checkout <NAME-OF-NEW-BRANCH>`
3. Edit the Puppetfile to track the modules and data needed in your new environment, and save your changes.
4. Commit your changes: `git commit -m "a commit message summarizing your change"`
5. Push your changes: `git push origin <NAME-OF-NEW-BRANCH>`
6. Deploy your environments as you normally would, either on the command line or with a Code Manager webhook.

**Delete an environment with code management**

To delete an environment with Code Manager or r10k, delete the corresponding branch from your control repository.

1. On the production branch of your control repo directory, on the command line, delete the environment's
   corresponding remote branch by running `git push origin --delete <BRANCH-TO-DELETE>`
2. Delete the local branch by running `git branch -d <BRANCH-TO-DELETE>`
3. Deploy your environments as you normally would, either on the command line or with a Code Manager webhook.

**Note:** If you use Code Manager to deploy environments with the webhook, deleting a branch from your control
repository does not immediately delete that environment from the master's live code directories. Code Manager will
delete the environment when it next deploys changes to any other environment. Alternately, to delete the environment
immediately, deploy all environments manually, run `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. You'll 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 (&#47;modules) in the same directory the Puppetfile is in. For example, declaring the puppetlabs-apache module in the Puppetfile normally installs the module into &quot;/modules/apache&quot;.

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, &quot;/modules&quot; relative to the location of the Puppetfile.)

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 your text editor, edit the Puppetfile, declaring modules and data content for your environment.

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.

After creating your Puppetfile, 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 should keep 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.

```puppet
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 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.

```puppet
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 puppetlabs-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.

For Bitbucket, you must preface your URL with `ssh://`, such as `ssh://git@git.example.com:site_data.git`

**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.

```puppet
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',
  :install_path => ''
```

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'
   ```

2. Configure the correct private key by setting 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.

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 the following options. Setting these options maintains the repository at that version and deploys any updates made 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 repo by a certain tag value.
   • `commit`: Specifies the repo by a certain commit.
   • `branch`: Specifies a certain branch of the repo.
   • `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 repo.

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.
To track a branch within the control repo, declare the content with the :branch option set to :control_branch.

```puppet
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 should deploy 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.

```puppet
# 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. Push code updates to your source control repo, and then Puppet syncs the code to your masters, so that all your servers start running the new code at the same time, without interrupting agent runs.

- **How Code Manager works** on page 574
  Code Manager uses r10k and the file sync service to stage, commit, and sync your code, automatically managing your environments and modules.

- **Configuring Code Manager** on page 575
  To configure Code Manager, you'll enable it in Puppet Enterprise (PE), set up authentication, and test the communication between the control repo and Code Manager.

- **Customize Code Manager configuration in Hiera** on page 580
  To customize your Code Manager configuration, set parameters with Hiera.

- **Triggering Code Manager on the command line** on page 587
  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 592
  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 594
  Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repos, or custom notifications.

- **Troubleshooting Code Manager** on page 595
  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 598
  Use Code Manager endpoints to deploy code and query environment deployment status on your masters without direct shell access.

**How Code Manager works**

Code Manager uses r10k and the file sync service to stage, commit, and sync your code, automatically managing your environments and modules.

First, you'll create a control repository with branches for each environment that you want to create (such as production, development, or testing). You'll also create a Puppetfile for each of your environments, specifying exactly which modules to install in each environment. This allows Code Manager to create directory environments, based on the branches you've set up. When you push code to your control repo, you'll trigger Code Manager to pull that new code into a staging code directory (`/etc/puppetlabs/code-staging`). File sync then picks up those changes, pauses Puppet Server to avoid conflicts, and then syncs the new code to the live code directories on your masters.

For more information about using environments in Puppet, see About Environments.

**Understanding file sync and the staging directory**

To sync your code across multiple masters 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`. But the file sync service looks for code in a code staging directory (`/etc/puppetlabs/code-staging`), and then syncs that to the live codedir.

Code Manager moves new code from source control into the staging directory, and then file sync moves it into the live code directory. This means you no longer write code to the codedir; if you do, the next time Code Manager deploys code from source control, it overwrites your changes.

For more detailed information about how file sync works, see the related topic about file sync.
CAUTION:
Don't directly modify code in the staging directory. Code Manager overwrites it with updates from the control repo. Similarly, don't modify code in the live code directory, because file sync overwrites that with code from the staging directory.

Related information
About file sync on page 620
File sync helps Code Manager keep your Puppet code synchronized across multiple masters.

Environment isolation metadata and Code Manager
Both your live and staging code directories contain metadata files that are generated by file sync to provide environment isolation for your resource types.

These files, which have a .pp extension, ensure that each environment uses the correct version of the resource type. Do not delete or modify these files. Do not use expressions from these files in regular manifests.

These files are generated as 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 will be generated as you make and deploy changes to your existing environments.

For more details about these files and how they isolate resource types in multiple environments, see environment isolation.

Moving from r10k to Code Manager
Switching from r10k to Code Manager can improve automation of your code management and deployments, but some r10k users might not be ready to switch yet.

Code Manager uses r10k in the background to improve automation of your code management and deployment. However, switching to Code Manager means you can no longer use r10k manually. Because of this, some r10k users might not want to move to Code Manager yet. Specifically, be aware of the following restrictions:

• If you use Code Manager, you cannot deploy code manually with r10k. If you depend on the ability to deploy modules directly from r10k, using the r10k deploy module command, you should continue to use your current r10k workflow.
• Code Manager must deploy all control repos to the same directory. If you are using r10k to deploy control repos to different directories, you should continue to use your current r10k workflow.
• Code Manager 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, or any other r10k configuration that Code Manager doesn't yet support, you should continue to use your current r10k workflow. If you are using a custom script to deploy code, you should carefully assess whether Code Manager meets your needs.

Related information
Managing code with r10k on page 608
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.

Configuring Code Manager
To configure Code Manager, you'll enable it in Puppet Enterprise (PE), set up authentication, and test the communication between the control repo and Code Manager.

When you've finished configuration, you'll be ready to deploy environments with Code Manager.

You can enable and configure Code Manager either during or after r10k installation.

To enable Code Manager after a new installation or in an existing PE installation, you'll set Code Manager parameters in the console. You can also configure Code Manager during a fresh PE installation, but only during a text-mode installation.
1. Enable and configure Code Manager, after installation or upgrade, by setting parameters in the master profile class in the PE console. Alternatively, enable during a fresh installation, by setting parameters in `pe.conf`.

2. Test your control repo.

3. Set up authentication for Code Manager.

4. Test Code Manager.

**Important:** If you enable Code Manager, do not attempt to follow the workflows in the PE getting started guides. The `puppet module` command is not compatible with Code Manager.

**Upgrading from r10k to Code Manager**

If you are upgrading from r10k to Code Manager, you must first disable your old r10k installation.

If you are upgrading from r10k to Code Manager, check the following before enabling Code Manager:

- If you used r10k prior to PE 2015.3, you might have configured r10k in the console using the `pe_r10k` class. If so, you must remove the `pe_r10k` class in the console before configuring Code Manager.
- If you used any previous versions of r10k, disable any tools that might automatically run it. Most commonly, this is the `zack-r10k` module. Code Manager cannot install or update code properly if other tools are running r10k.

When you start using Code Manager, it runs r10k in the background. You can no longer directly interact with r10k or use the `zack-r10k` module.

**Enable Code Manager**

Usually, you enable Code Manager after PE is already installed. If you are automating your PE installation and using an existing control repo and SSH key, you can enable Code Manager during the PE installation process.

**Before you begin**

- Your SSH key should be generated without setting a passphrase.

- The private key file must be located on the master, owned by the user, and located in a directory that the user has permission to view. We recommend `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`.

**Enable Code Manager after installation**

To enable Code Manager after installing PE or in an existing installation, set parameters in the console.

1. In the console, set the following parameters in the `puppet-enterprise::profile::master` class in the PE Master node group.

   - `code_manager_auto_configure` to `true`: This enables and configures both Code Manager and file sync.

   - `r10k_remote`: This is the location of your control repository, as accessed by SSH. Enter a string that is a valid SSH URL for your Git control repository. For example: `"git@YOUR.GIT.SERVER.COM::puppet/control.git"`.

      **Note:** Some Git providers, such as Bitbucket, might have additional requirements for enabling SSH access. See your provider's documentation for information.

   - `r10k_private_key`: Enter a string specifying the path to the SSH private key that permits the user to access your Git repositories.
2. Run Puppet on all masters.

If you run Puppet for all your masters at the same time, such as with Run Puppet in the console, you might see errors like this your compilers' logs:

```
```

You can ignore these errors. They occur because Puppet Server is restarting while the compile masters are trying to poll for new code. These errors should stop as soon as the Puppet Server on the master of masters has finished restarting.

Next, set up authentication.

**Enable Code Manager during installation**

To configure Code Manager during a fresh installation of PE, add parameters to the `pe.conf` file.

Use these parameters **only** with text-mode PE installation, not with web-based installation. Adding the listed parameters enables and configures file sync and Code Manager.

1. Add the following three parameters to `pe.conf` *before* installation, adding your specific URL and directory information:

   ```
   "puppet_enterprise::profile::master::r10k_remote": "git@<YOUR.GIT.SERVER.COM>:puppet/control.git"
   
   "puppet_enterprise::profile::master::r10k_private_key": "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"
   
   puppet_enterprise::profile::master::code_manager_auto_configure": true
   ```

   These parameters specify the private key location and the control repo URL, and enables Code Manager and file sync.
   - "puppet_enterprise::profile::master::r10k_remote"
     
     This setting specifies the location of the control repository. It accepts a string that is a valid URL for your Git control repository.

     **Note:** For Bitbucket, you must preface your remote URL with `ssh://`, to match the format of Bitbucket’s SSH URLs. For example, `ssh://git@github.domain.com/pup/puppet.git`.
   - puppet_enterprise::profile::master::r10k_private_key
     
     This setting accepts a string that specifies the path to the future location of the SSH private key used to access your Git repositories. **After** PE installation is completed, place the key in the specified location. You do this in step 3 below.
   - puppet_enterprise::profile::master::code_manager_auto_configure
     
     This setting configures Code Manager, the Git control repository to use for storing code, and the private key for accessing your Git repos. Accepts the values `true` or `false`. Set it to `true` to enable and configure Code Manager. A `false` setting disables Code Manager and file sync.

2. Complete the installation, following the steps in the text-based installation instructions.

3. After installation is complete, place the SSH private key you created when you set up your control repository in the `r10k_private_key` location.

Next, set up authentication.
Related information
Install using text mode (mono configuration) on page 174
When you run the installer in text mode, you provide a configuration file (pe.conf) to the installer. The pe.conf file contains values for the parameters needed for installation.

Install using text mode (split configuration) on page 174
When you run the installer in text mode, you provide a configuration file (pe.conf) to the installer. The pe.conf file contains values for the parameters needed for installation.

Configuration parameters and the pe.conf file on page 178
A pe.conf file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

Set up authentication for Code Manager
To securely deploy environments, Code Manager needs an authentication token for both authentication and authorization.

Before you begin
Configure the Puppet access command line tool.
To generate a token for Code Manager, you'll first assign a user to the deployment role, and then request an authentication token.

Related information
Configuring puppet-access on page 290
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 Puppet master.

Assign a user to the deployment role
To request an authentication token, you must first assign a user the correct permissions with role-based access control (RBAC).

1. In the console, create a deployment user. We recommend that you create a dedicated deployment user for Code Manager use.
2. Add the deployment user to the Code Deployers role. This role is automatically created on install, with default permissions for code deployment and token lifetime management.

Next, request the authentication token.
Related information
Add a user to a user role on page 282
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.

Assign a user group to a user role on page 288
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.

Request an authentication token for deployments
Request an authentication token for the deployment user to enable secure deployment of your code.

By default, authentication tokens have a five-minute 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.

Generate the authentication token using the puppet-access command.

1. From the command line on the master, 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 specifying 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 username 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`. Next, test the connection to the control repo.

**Related information**

*Setting a token-specific lifetime* on page 293
Tokens have a default lifetime of five minutes, 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 290
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 control repo**
To make sure that Code Manager can connect to the control repo, test the connection to the repository.

From the command line, run `puppet-code deploy --dry-run`

For a more detailed list of what environments Code Manager found, run:

```
puppet-code deploy --log-level=info --dry-run
```

- If the control repo is set up properly, this command returns how many environments Code Manager found.
- If an environment is not set up properly or causes an error, it will not appear in the result.

**Test Code Manager**
To test whether Code Manager deploys your environments correctly, trigger a single environment deployment on the command line.

**Deploy a single environment**
Test Code Manager by deploying a single test environment.

This deploys the test environment, and then returns deployment results with the SHA (a checksum for the content stored) for the control repo commit.

From the command line, deploy one environment by running `puppet-code deploy my_test_environment --wait`

Check to make sure the environment was deployed. If so, you've set up Code Manager correctly.

If the deployment does not work as you expect, check over the configuration steps, or refer to the troubleshooting guide for help.

After Code Manager is fully enabled and configured, you can trigger it to deploy your environments.

There are several ways to trigger deployments, depending on your needs.

- Manually, on the command line.
- Automatically, with a webhook.
- Automatically, with a custom script that hits the deploys endpoint.

**Code Manager console settings**
After Code Manager is configured, you can adjust some settings in the master profile in the console.

These options are required for Code Manager to work, unless otherwise noted.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>code_manager_auto_config</td>
<td>Set to true to auto-configure Code Manager.</td>
<td>true</td>
</tr>
<tr>
<td>r10k_remote</td>
<td>The location of the Git control repository. Enter a string that is a valid URL for your control repository.</td>
<td>'git8&lt;YOUR.GIT.SERVER.COM&gt;:puppet/control.git'</td>
</tr>
</tbody>
</table>
To further customize your Code Manager configuration with Hiera, see the related topic about customizing your configuration.

**Customize Code Manager configuration in Hiera**

To customize your Code Manager configuration, set parameters with Hiera.

Add Code Manager parameters in the puppet_enterprise::master::code_manager class with Hiera.

**Important:**

Do not edit the Code Manager configuration file manually. Puppet manages this configuration file automatically and will undo any manual changes you make.

**Related information**

Configure settings with Hiera on page 236

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.

**Add Code Manager parameters in Hiera**

To customize your Code Manager configuration, add parameters to your control repository hierarchy in the hieradata/common.yaml file.

1. Add the parameter to hieradata/common.yaml in the format:
   puppet_enterprise::master::code_manager::<parameter>

   puppet_enterprise::master::code_manager::deploy_pool_size: 4
   puppet_enterprise::master::code_manager::timeouts_deploy: 300

   The first parameter in this example increases the size of the default worker pool, and the second reduces the maximum time allowed for deploying a single environment.

2. Run Puppet on the master to apply changes.

**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.

puppet_enterprise::master::code_manager::post_environment_hooks:
```yaml
  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": "6939889b679f21449545c44f26aa06174d25c21",
        "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1"},
    "environment": "production",
    "id": 3,
    "status": "complete"
}
```

**use-client-ssl**

Specifies whether the client SSL configuration should be used for HTTPS connections. If the hook destination is
a server using the Puppet CA, then this should be set to true; otherwise, it should be set 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 Forge settings**

To configure how Code Manager downloads modules from the Forge, specify `forge_settings` in Hiera.

This parameter configures where modules should be installed from, and sets a proxy for all interactions. The `forge_settings` parameter accepts a hash with the following values:

- baseurl
- proxy

**baseurl**

Indicates where modules should be installed from. Defaults to `https://forgeapi.puppetlabs.com`.

```
puppet_enterprise::master::code_manager::forge_settings:
```

© 2020 Puppet, Inc.
baseurl: 'https://private-forge.mysite'

proxy
Sets the proxy for all interactions.
This setting overrides the global proxy setting on 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.

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

private-key
Specifies the file containing the default private key used to access control repositories, such as /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa. 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 by the master profile in the console.

proxy
Sets a proxy specifically for operations that use an HTTP(S) transport.
This setting overrides the global proxy setting on 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 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 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.
• 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: "ssh://jfkennedy@gitserver.puppetlabs.net/repositories/repo_one.git"
    private-key: "/test_keys/jfkennedy"
- remote: "ssh://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 or operations only, set the proxy subsetting under the git_settings or forge_settings parameters. To override for a specific repository, set a proxy in the repositories list of git_settings. To override this setting with no proxy for , , 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 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 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

```
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 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".

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 will be named "myorg-main-modules", and the second will be "testing-main-modules".

```ini
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>remote</td>
<td>The location of the Git control repository. Either the remote or sources parameters must be specified. If remote is already specified in the master profile, that value is used here. If both the sources and remote keys are specified, the sources key overrides remote.</td>
<td>A string that is a valid SSH URL for your Git remote.</td>
<td>No default.</td>
</tr>
<tr>
<td>authenticate_webhook</td>
<td>Turns on RBAC authentication for the /v1/webhook endpoint.</td>
<td>Boolean.</td>
<td>true</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>Defaults to $::clientcert.</td>
</tr>
<tr>
<td>data</td>
<td>The path to the directory where Code Manager should store internal file content.</td>
<td>A valid file path.</td>
<td>/opt/puppetlabs/server/data/code-manager</td>
</tr>
<tr>
<td>deploy_pool_size</td>
<td>Specifies the number of threads in the worker pool; this dictates how many deploy processes can run in parallel.</td>
<td>An integer.</td>
<td>2</td>
</tr>
<tr>
<td>deploy_ttl</td>
<td>Specifies the length of time completed deployments will be retained before garbage collection. For usage details, see configuring garbage collection.</td>
<td>The time as a string, using any of the following suffixes:</td>
<td>1h (one hour)</td>
</tr>
<tr>
<td></td>
<td>• d - days</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• h - hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• m - minutes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• s - seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>hostcrl</td>
<td>The path to the SSL CRL.</td>
<td>A valid file path.</td>
<td>$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>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</td>
<td>An integer.</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>respond. Used for both the socket connect timeout and the read timeout, so</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the longest total timeout would be twice the specified value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeouts_shutdown</td>
<td>Maximum time (in seconds) to wait for in-progress deploys to complete when</td>
<td>An integer.</td>
<td>610</td>
</tr>
<tr>
<td></td>
<td>shutting down the service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeouts_shutdown</td>
<td>Maximum time that a request sent with a wait parameter should wait for</td>
<td>An integer.</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td>each environment before timing out.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeouts_sync</td>
<td>Maximum time (in seconds) that a request sent with a wait parameter should</td>
<td>An integer.</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>wait for all compile masters to receive deployed code before timing out.</td>
<td></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>
<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 <code>file_sync_auto_commit</code> is set to <code>true</code>, this defaults to <code>/etc/puppetlabs/code-staging/environments</code>. See <code>file_sync_auto_commit</code>.</td>
<td>Directory</td>
<td><code>/etc/puppetlabs/code-staging/environments</code></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>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. • 'error': Ignores branches with non-word characters and issues an error.</td>
<td>'error'</td>
</tr>
<tr>
<td>git_settings</td>
<td>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.</td>
<td>Accepts a hash of:</td>
<td>Default private-key value as set in console. No other default settings.</td>
</tr>
</tbody>
</table>
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
</tr>
<tr>
<td>sources</td>
</tr>
</tbody>
</table>

### Description

- See [Configuring Git settings](#) for usage details.
- Configures a proxy server to use for all operations that occur over an HTTP(S) transport. See [Configuring proxies](#) for usage details.
- 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.

### Value

- Accepts:
  - A proxy server.
  - An authenticated proxy with Basic or Digest authentication.
  - An empty value.
- A hash of:
  - remote
  - prefix

### Default

- No default.
- 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 masters 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:

```json
{
    "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
    "token-file": "~/.puppetlabs/token",
    "service-url": "https://<PUPPET MASTER HOSTNAME>:8170/code-manager"
}
```

**Important:**

On a PE-managed machine, Puppet manages this file for you. Do not manually edit this file, because Puppet will overwrite 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`. © 2020 Puppet, Inc.
The default configuration file looks something like:

```json
{
  "cacert": "C:\\ProgramData\\PuppetLabs\\puppet\\etc\\ssl\\certs\\ca.pem",
  "token-file": "C:\\Users\\<username>\\.puppetlabs\\token",
  "service-url": "https://<PUPPET MASTER 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 config 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 compile masters, 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.

```bash
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 590

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 will be 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://<PUPPET MASTER 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://<PUPPET MASTER HOSTNAME>:8170/code-manager"
}
```

**Related information**

Installing PE client tools on page 213
PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that is not necessarily 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 master, 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 <code>puppet-code</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 <code>puppet-code.conf</code> file that takes precedence over all other existing <code>puppet-code.conf</code> files.</td>
<td>A path to a <code>puppet-code.conf</code> 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>Option</td>
<td>Description</td>
<td>Accepted arguments</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>--token-file, -t</td>
<td>Sets an authentication token that overrides the <code>token-file</code> 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 <code>service-url</code> 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.</td>
<td>Accepts either the name of a single environment or the <code>--all</code> flag.</td>
<td><code>--wait, -w; --all; --dry-run; --format, -F</code></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.</td>
<td>Accepts log levels none, trace, info, warn, error, fatal.</td>
<td>none</td>
</tr>
</tbody>
</table>

**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 compile masters, and  
4. Return the deployment signature with control repo 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`. |
### Managing and deploying Puppet code

#### 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>cacert</td>
<td>Specifies the path to the Puppet CA certificate to use when connecting to the Code Manager service over SSL.</td>
<td><code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code></td>
<td><code>C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem</code></td>
</tr>
<tr>
<td>token-file</td>
<td>Specifies the location of the file containing the authentication token for Code Manager.</td>
<td><code>~/.puppetlabs\token</code></td>
<td><code>C:\Users\&lt;username&gt;\.puppetlabs\token</code></td>
</tr>
</tbody>
</table>

#### 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'll need to 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 Puppet master 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`).
• The complete authentication token you generated earlier (token=<TOKEN>), passed with the token query parameter. See requesting an authentication token in the configuration documentation.

For example, the URL for a GitHub webhook might look like this:

https://code-manager.example.com:8170/code-manager/v1/webhook?
type=github&token=<TOKEN>

The URL for a Stash webhook might look something like this:

https://code-manager.example.com:8170/code-manager/v1/webhook?
type=stash&prefix=dev&token=<TOKEN>

With the complete token attached, a GitHub URL looks something like this:

https://code-manager.example.com:8170/code-manager/v1/webhook?
type=github&token=0WJ4YPJVyQz26xm3X2110ihb7MUa6812CZWjxM3vt4mQ

**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
  • stash: Bitbucket Server (Stash)
  • bitbucket: Bitbucket
  • tfs-git: Team Foundation Server (resource version 1.0 is supported)
• **prefix**: Specifies a prefix for converting branch names to environments.
• **token**: Specifies the entire PE authorization token.

**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 repos, 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 578
Request an authentication token for the deployment user to enable secure deployment of your code.

Code Manager API on page 598
Use Code Manager endpoints to deploy code and query environment deployment status on your masters 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 587
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/deploy endpoint.

To deploy environments with a `curl` command in your custom script, compose a `curl` command to hit the `/v1/deploy` 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 Puppet master server name (for example, `master.example.com`)

© 2020 Puppet, Inc.
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://master.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 `cat ~/.puppetlabs/token` in the body of your request. For example:

  ```sh
  ```
  
  Attach the entire token to your Code Manager URL using the `token` query parameter. For example:

  ```sh
  https://master.example.com: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>pe-puppetserver process isn’t running</td>
</tr>
<tr>
<td>Code Manager reports invalid configuration</td>
<td>Invalid configuration at /etc/puppetlabs/puppetserver/conf.d/code-manager.conf</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 master 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:

```bash
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 master or master of masters 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'll need a copy of the Puppetfile in a temporary directory.

Create a temporary directory `/var/tmp/test-puppetfile` on the master 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:

```bash
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 will work 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 should be used only for ad-hoc testing.
Run this deployment test with the following command:

```bash
sudo -H -u pe-puppet bash -c \
'/opt/puppetlabs/puppet/bin/r10k deploy environment -c /opt/puppetlabs/\nserver/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 will 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:

```bash
curl -k -X POST -H 'Content-Type: application/json' \
-H "X-Authentication: `cat ~/.puppetlabs/token`" \
https://<URL.OF.CODE.MANAGER.SERVER>:8170/code-manager/v1/deploys \n-d '{"environments": ["<ENV_NAME>"], "wait": true}'
```

This request waits for the deployment and sync to compile masters to complete ("wait": true) and so returns errors from the deployment.

Alternatively, you can deploy all environments with the following curl command:

```bash
curl -k -X POST -H 'Content-Type: application/json' \
-H "X-Authentication: `cat ~/.puppetlabs/token`" \
https://<URL.OF.CODE.MANAGER.SERVER>:8170/code-manager/v1/deploys \n-d '{"deploy-all": true, "wait": true}'
```
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 your `timeouts_deploy` key. You might also need to increase `timeouts_shutdown`, `timeouts_sync`, and `timeouts_wait`.

**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 masters 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:

```
```

Passing the token with the `token` query parameter might look like:

```
```

**Related information**
Request an authentication token for deployments on page 578

© 2020 Puppet, Inc.
Request an authentication token for the deployment user to enable secure deployment of your code.

**POST /v1/deploy**

Use the Code Manager /deploy endpoint to queue code deployments.

**Request format**

The body of the POST /deploy request must be a JSON object that includes the authentication token and a URL containing:

- The name of the master server, such as master.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 a comma- and space-separated list of strings specifying environments.</td>
</tr>
<tr>
<td>&quot;deploy-all&quot;</td>
<td>Specifies whether Code Manager should deploy all environments. 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 &quot;environments&quot; key must be specified with a list of environments.</td>
<td>true, false</td>
</tr>
<tr>
<td>&quot;wait&quot;</td>
<td>Specifies whether Code Manager should wait for all deploys to finish or error before it returns a response. If specified true, Code Manager returns a response after all deploys have either finished or errored. If specified false or not specified, returns a response showing that deploys are queued. Specify together with the &quot;deploy-all&quot; or &quot;environment&quot; keys.</td>
<td>true, false</td>
</tr>
<tr>
<td>&quot;dry-run&quot;</td>
<td>Tests the Code Manager connection to each of the configured remotes and attempts to fetch a list of environments from each, reporting any connection errors.</td>
<td>true, false</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:

```bash
```

To deploy all configured environments and return only the queuing results, pass the "deploy-all" key without the "wait" key:

```bash
```

**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>Definition</th>
<th>Values</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 masters.</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>
</tbody>
</table>
Key | Description | Value
---|---|---
"deploy-signature" | The commit SHA of the control repo that Code Manager used to deploy code in that environment. | A Git SHA.
"file-sync" | Commit SHAs used internally by file sync to identify the code synced to the code staging directory | Git SHAs for the "environment-commit" and "code-commit" keys.

For example, for a /deploy 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.

```json
[
  {
    "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:

```json
[
  {
    "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
    "file-sync": {
      "environment-commit": "6939889b679fd4b449545c44f26aa06174d25c21",
      "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1"
    },
    "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-env-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>Key</td>
<td>Definition</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</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-env-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/puffpetlabs-apache".
},
    "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 Puppet master server (for example, master.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>
<tbody>
<tr>
<td>type</td>
<td>Required. Specifies what type of POST body to expect.</td>
<td>github, gitlab, tfs-git, bitbucket, stash</td>
</tr>
<tr>
<td>prefix</td>
<td>Optional. Specifies a prefix to use in translating branch names to environments.</td>
<td>A string specifying a branch prefix.</td>
</tr>
<tr>
<td>token</td>
<td>Specifies the PE authorization token to use. Required unless you disable</td>
<td>The complete authentication token.</td>
</tr>
</tbody>
</table>
For example, a GitHub webhook might look like this:

```
https://master.example.com:8170/code-manager/v1/webhook?type=github&token=$TOKEN
```

A Stash webhook with the optional prefix parameter specified might look like:

```
https://master.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://master.example.com:8170/code-manager/v1/webhook?
type=github&token=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkZXBsb3NzcyIsImlhdCI6MTQ3ODI4Njk5MCwiZXhwIjoxNDY4MTg5MzQ3LCJpYXQiOjE0MDQyNjEyMjEsImlzY29yZCI6ImV4cGVocmFjdFwiLCJzdWJwYXJkb3giOiJnYW5vbGVvbl9wYXJkb3giLCJzdWIiOiJ1c2VyIm0iLCJpZCI6IjEzMmU0Y2Y0Y2U1YiIsInVzZXJfaWQiOiJ0cmFjdFwiLCJ0eXBlIjoiQWRvYmUgSW1haWwgaXJzd29yZCBHSSBub29sdGljZSBNYXJjaGVyIn0.OSdEIA-HGRC0r2J87P2JsDsyz-EMK-7sz2alTswy2e3uSv1Z6ulXaIQd69PqNSSBExQtExDwXZlNyQa_le2gwTo4smjUaBd6_lnPYr6Gj4hB4-fT8dNNvVv2aESWFkt-nsNKJwE9LiEd4W42aCYfse1KNgPuXR5m75SrsUy86yKhlzVFHKqqexEyuS7LGeQJvyQE1qEejSdbiLg6zn1JXhglWzWz5p2fNrx1oxrrkU02xioUOdegGyCwvNiaMazztM9NyDdWmZ4dKjsqMosu0CRkMswcYPPaeIcsYFI7XsaeC65N4RLKhUfwiXXE-uODeHcl3mTr9rwZGmU3WrY7t6w1bDM9FomPejGM2aJoZ05PinAIYvX3oH5QJ9famO0pVlb-m13bvkKm2wKagpc4DhIut9sqLMagG8-AwMA4r_0EvLwFzf8c1zk343zNyPG7BvlnPle99HjQues690L-fknSdfXyRZerTHvZ0pSjWzvUSR49etmK-0xnMbQEq4tCBW2r_khEG5jUDzeKt3PIiXdxmUaaEPHo6V19XIY5
```

## 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 will 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
Use the Code Manager /deploys/status endpoint to return the status of the code deployments that Code Manager is processing for each environment.

Request format
The body of the GET /deploys/status request must include the authentication token and a URL containing:

- The name of the master server, such as master.example.com.
- The Code Manager port, such as 8170.
- The endpoint.

```bash
```

Response format
The /deploys/status endpoint responds with a list of the 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 "queued", "deploying", "new", 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 each master that Code Manager is deploying environments to, including whether the code in the master'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 environment:</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 environment:</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;new&quot;</td>
<td>Code Manager has accepted a request to deploy the environment, but has not yet queued it.</td>
<td>For each environment:</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 environment:</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-env-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>Key</td>
<td>Definition</td>
<td>Values</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>&quot;environment&quot;</td>
<td>The name of the environment.</td>
<td>A string that is the name of the environment.</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>
<tr>
<td>&quot;deployed&quot;</td>
<td>Lists information for each deployed environment.</td>
<td>For each environment:</td>
</tr>
<tr>
<td></td>
<td>• &quot;environment&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;queued-at&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;date&quot;</td>
<td>The date and time the operation was completed.</td>
<td>A date and time stamp.</td>
</tr>
<tr>
<td>&quot;deploy-signature&quot;</td>
<td>The commit SHA of the control repo used to deploy the environment.</td>
<td>A Git SHA.</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 masters.</td>
<td>true, false</td>
</tr>
<tr>
<td>&quot;file-sync-clients&quot;</td>
<td>List of all masters that Code Manager is deploying code to.</td>
<td>For each environment listed:</td>
</tr>
<tr>
<td></td>
<td>• &quot;last_check_in_time&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;synced-with-file-sync-storage&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;deployed&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;environment&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;date&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &quot;deploy-signature&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;last_check_in_time&quot;</td>
<td>The most recent time that the live code directory checked in for new code from the staging directory.</td>
<td>A date and time stamp.</td>
</tr>
<tr>
<td>&quot;synced-with-file-sync-storage&quot;</td>
<td>Whether the live code directory has been synced with the code staging directory on a given master.</td>
<td>true, false</td>
</tr>
</tbody>
</table>

For example, a complete response to a request might look like:

```json
{
  "deploys-status":{
    "queued":[
      {
        "environment":"dev",
        "queued-at":"2018-05-15T17:42:34.988Z"
      },
      {
        "environment":"test",
        "queued-at":"2018-05-15T17:42:34.988Z"
      },
      {
        "environment":"prod",
```
"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",
          "deploy-signature":"66d620604c9465b464a3dac4884f96c43748b2c5"
        },
        { "environment":"test",
          "date":"2018-05-11T22:40:48.000Z",
          "deploy-signature":"24ecc7bac8a4d727d6a3a2350b6fda53812ee86f"
        },
        { "environment":"dev",
          "date":"2018-05-11T22:40:50.000Z",
          "deploy-signature":"503a335c99a190501456194d13ff722194e55613"
        }
      ]
    }
  }
}
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-env-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:

```json
[
  {
    "deploys-status": {
      "deploying": [],
      "failed": [
        {
          "environment": "test14",
          "error": {
            "details": {
              "corrected-env-name": "test14",
            },
            "kind": "puppetlabs.code-manager/deploy-failure",
            "msg": "Errors while deploying environment 'test14' (exit code: 1):
            Authentication failed for Git remote "https://github.com/puppetlabs/puppetlabs-apache"
          },
        },
        "new": [],
        "queued": []
      ],
      "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.

Based on the code in your control repo branches, r10k creates environments on your master and installs and updates the modules you want in each environment. If you are running PE 2015.3 or later, we encourage you to use Code Manager, which uses r10k in the background to automate the deployment of your code. If you use Code Manager, you won't need to manage or interact with r10k manually. To learn more about Code Manager and begin setup, see the Code Manager page. However, if you're already using r10k and aren't ready to switch to Code Manager, you can continue using r10k alone.

To set up r10k to manage your environments:

1. Set up a control repository for your code.
2. Create a Puppetfile to manage the content for your environment.
3. Configure r10k. Optionally, you can also customize your r10k configuration in Hiera.
4. Run r10k to deploy your environments and modules.

We've also included a reference of r10k subcommands.

- Configuring r10k on page 608
  When performing a fresh text-mode installation of Puppet Enterprise (PE), you can configure r10k by adding parameters to the `pe.conf` file. In existing installations, configure r10k by adjusting parameters in the console.

- Customizing r10k configuration on page 610
  If you need a customized r10k configuration, you can set specific parameters with Hiera.

- Deploying environments with r10k on page 616
  Deploy environments on the command line with the `r10k` command.

- r10k command reference on page 618
  r10k accepts several command line actions, with options and subcommands.

Configuring r10k

When performing a fresh text-mode installation of Puppet Enterprise (PE), you can configure r10k by adding parameters to the `pe.conf` file. In existing installations, configure r10k by adjusting parameters in the console.

⚠️ CAUTION: Do not edit the r10k configuration file manually. Puppet manages this configuration file automatically and will undo any manual changes you make.

Related information

Install using text mode (mono configuration) on page 174
When you run the installer in text mode, you provide a configuration file (`pe.conf`) to the installer. The `pe.conf` file contains values for the parameters needed for installation.

Configuration parameters and the `pe.conf` file on page 178
A `pe.conf` file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

Upgrading from previous versions of r10k

If you used r10k prior to PE version 2015.3, you might have configured it in the console using the `pe_r10k` class. We suggest configuring r10k in the master profile class, and then customizing your configuration as needed in Hiera.

This simplifies configuration, and makes it easier to move to Code Manager in the future.

To switch to master profile class configuration, remove the `pe_r10k` class in the console, and then configure r10k as described in the topic about configuring r10k after PE installation. You can then customize your configuration in Hiera if needed.

Note: If you were using earlier versions of r10k with the `zack-r10k` module, discontinue use of the module and switch to the master profile configuration as above.
Configure r10k during installation
To set up r10k during PE installation, add the r10k parameters to `pe.conf` before starting installation. This is the easiest way to set up r10k with a new PE installation.

**Before you begin**
Ensure that you have a Puppetfile and a control repo. You'll also need the SSH private key that you created when you made your control repo.

**Restriction:** This configuration method works only with text-based installation. If you are using the web-based installer, see the related topic about configuring r10k after PE installation.

1. Add `puppet_enterprise::profile::master::r10k_remote` to `pe.conf`.
   This setting specifies the location of the control repository. It accepts a string that is a valid URL for your Git control repository.

   "puppet_enterprise::profile::master::r10k_remote": "git@<YOUR.GIT.SERVER.COM>:puppet/control.git"

2. Add `puppet_enterprise::profile::master::r10k_private_key` to `pe.conf`.
   This setting specifies the path to the file that contains the SSH private key used to access your Git repositories. This location for the SSH private key file, which you created when you set up your control repository, must be located on the Puppet master and owned by and accessible to the pe-puppet user. The setting accepts a string.

   "puppet_enterprise::profile::master::r10k_private_key": "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"

3. Complete PE installation. The installer configures r10k for you. You can change the values for the remote and the private key as needed in the master profile settings in the console.

4. After PE installation is complete, place the SSH private key you created when you set up your control repository in the `r10k_private_key` location.

5. Run r10k. PE does not automatically run r10k.

Configure r10k after PE installation
To configure r10k in an existing PE, set r10k parameters in the console. You can also adjust r10k settings in the console.

**Before you begin**
Ensure that you have a Puppetfile and a control repo. You'll also need the SSH private key that you created when you made your control repo.

1. In the console, set the following parameters in the `puppet_enterprise::profile::master` class in the PE Master node group:
   • `r10k_remote`  
     This is the location of your control repository. Enter a string that is a valid URL for your Git control repository, such as "git@<YOUR.GIT.SERVER.COM>:puppet/control.git".
   • `r10k_private_key`  
     This is the path to the private key that permits the pe-puppet user to access your Git repositories. This file must be located on the Puppet master, owned by the pe-puppet user, and in a directory that the pe-puppet user has permission to view. We recommend `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`. Enter a string, such as "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa".

2. Run Puppet on all of your masters.
You can now customize your r10k configuration in Hiera, if needed. After r10k is configured, you can deploy your environments from the command line. PE does not automatically run r10k at the end of installation.

**Customizing r10k configuration**

If you need a customized r10k configuration, you can set specific parameters with Hiera.

To customize your configuration, add keys to your control repository hierarchy in the `hieradata/common.yaml` file in the format `pe_r10k::<parameter>`.

In this example, the first parameter specifies where to store the cache of your Git repos, and the second sets where the source repository should be fetched from.

```yaml
pe_r10k::cachedir: /var/cache/r10k
pe_r10k::remote: git://git-server.site/my-org/main-modules
```

**Remember:** After changing any of these settings, run r10k on the command line to deploy your environments. PE does not automatically run r10k at the end of installation.

**Related information**

Configure settings with Hiera on page 236

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.

**Add r10k parameters in Hiera**

To customize your r10k configuration, add parameters to your control repository hierarchy in the `hieradata/common.yaml` file.

1. Add the parameter in the `hieradata/common.yaml` file in the format `pe_r10k::<parameter>`.
2. Run Puppet on the master to apply changes.

**Configuring Forge settings**

To configure how r10k downloads modules from the Forge, specify `forge_settings` in Hiera.

This parameter configures where modules should be installed from, and sets a proxy for all interactions. The `forge_settings` parameter accepts a hash with the following values:

- `baseurl`
- `proxy`

**baseurl**

Indicates where modules should be installed from. Defaults to `https://forgeapi.puppetlabs.com`.

```yaml
pe_r10k::forge_settings:
  baseurl: 'https://private-forge.mysite'
```

**proxy**

Sets the proxy for all interactions.

This setting overrides the global `proxy` setting on 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
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 will purge during code deployments. Available values are:

- deployment
- environment
- puppetfile
- purge_whitelist

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.

```python
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 --puppetfile 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 --puppetfile 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_whitelist

The purge_whitelist 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*.

© 2020 Puppet, Inc.
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.

```python
deploy:
    purge_whitelist: [ '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.

```python
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`

```yaml
myorg:
    remote: "git://git-server.site/myorg/main-modules"
    basedir: "/etc/puppetlabs/puppet/environments"
    prefix: true
    invalid_branches: 'error'

mysource:
    remote: "git://git-server.site/mysource/main-modules"
    basedir: "/etc/puppetlabs/puppet/environments"
    prefix: "testing"
    invalid_branches: 'correct_and_warn'
```

### 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 will be created for this source. This directory is entirely managed by r10k, and any contents that r10k did not put there will be 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 will be 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 will be named "myorg-main-modules", and the second will be "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 will have its name tested against all prefixes listed in `ignore_branch_prefixes` and, if the prefix is found, then an environment will not be deployed for this branch.

The setting is a list of strings. In the following example, branches in "mysource" with the prefixes "test" and "dev" will not be deployed.

```---
   sources:
     mysource:
       basedir: '/etc/puppet/environments'
       ignore_branch_prefixes:
         - 'test'
         - 'dev'
```  

If `ignore_branch_prefixes` is not specified, then all branches will be 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 will be created for this source.

This directory is entirely managed by r10k, and any contents that r10k did not put there will be 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 won't be able to 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:

- `private_key`
- `proxy`
- `repositories`
• provider

Contains settings specific to Git. Accepts a hash of values, such as:

```ruby
pe_r10k::git_settings:
  provider: "rugged"
  private_key: "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"
  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.rsa`. 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 by the master profile in the console.

**proxy**

Sets a proxy specifically for operations that use an HTTP(S) transport.

This setting overrides the global `proxy` setting on 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 repository only, set `proxy` in the `repositories` subsetting of `git_settings`. See the global `proxy` setting for more information and examples.

```ruby
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.

```ruby
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"
      proxy: "https://proxy.example.com:3128"
    - 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 or operations only, set the proxy subsetting under the git_settings or forge_settings parameters. To override for a specific repository, set a proxy in the repositories list of git_settings. To override this setting with no proxy for 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:

```bash
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. See Configuring purge levels and Locking deployments for usage details.</td>
<td>Accepts settings for:</td>
<td>• purge_level: ['deployment', 'puppetfile']</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• write_lock</td>
<td>• write_lock: Not set by default.</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>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: • baseurl • proxy</td>
<td>No default.</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: • provider • private_key • proxy • repositories</td>
<td>No default.</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>No default set.</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>No default set.</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>By default, uses the r10k_remote value set in the console.</td>
</tr>
<tr>
<td>r10k_basedir</td>
<td>Specifies the path where environments will be created for this source. See Configuring the base directory for usage details.</td>
<td>A valid file path, which must match the value of environmentpath in your puppet.conf file.</td>
<td>Uses the value of the environmentpath in your puppet.conf file.</td>
</tr>
<tr>
<td>sources</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: • basedir • remote • prefix • ignore_branch_prefixes</td>
<td>No default.</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 what 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.

**Note:**

When running commands to deploy code on a master, r10k needs write access to your environment path. Run r10k as the pe-puppet user or as root. Running the user as root requires access control to the root user.

**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.

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 updates only the environment itself.

From the command line, run `r10k deploy environment`

**Updating all environments and modules**

With the `--puppetfile` flag, the `deploy environment` command updates all environments and modules.

This command:

- Updates all sources.
- Creates new environments.
- Deletes old environments.
- 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 should use the less resource-intensive commands for updating environments and modules.

From the command line, run `r10k deploy environment --puppetfile`

**Updating a single environment**

To update just one environment, specify that environment name with the `deploy environment` command.

This command updates modules only on the first deployment of the specified environment. On subsequent updates, it updates only the environment itself.

If you're actively developing on a given environment, this is the quickest way to deploy your changes.

On the command line, run `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 `--puppetfile` 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> --puppetfile`

**Installing and updating modules**

To update modules, use the `r10k deploy module` subcommand. This command installs or updates the modules you've 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.

You can specify a single module or multiple modules. This is useful when you're working on a module and want to update it across all environments.
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.

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, `-p`. From the command line, run `r10k deploy display -p`.
- To get detailed information about module versions, use the `-p` (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 -p --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:

  ```bash
  r10k deploy display -p --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
r10k command options

r10k commands accept the following options.

<table>
<thead>
<tr>
<th>r10k command options</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>--color</td>
<td>Enables color coded log messages.</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows help for this command.</td>
</tr>
<tr>
<td>--trace, -t</td>
<td>Displays stack traces on application crash.</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Sets log verbosity. Valid values: fatal, error, warn, notice, info, debug, debug1, debug2.</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 --detail</td>
<td>Displays detailed information.</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 --p</td>
<td>Updates modules specified in the environment's Puppetfile.</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>

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>puppetfile subcommand</th>
<th>Action</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 --puppetfile</td>
<td>Installs modules from a custom Puppetfile path.</td>
</tr>
<tr>
<td>install --module-dir</td>
<td>Installs modules from a Puppetfile to a custom module directory location.</td>
</tr>
</tbody>
</table>
### About file sync

File sync helps Code Manager keep your Puppet code synchronized across multiple masters.

When triggered by a web endpoint, file sync takes changes from your working directory on your master of masters (MoM) and deploys the code to a live code directory. File sync then automatically deploys that code onto all your compile masters, ensuring that all masters in a multi-master configuration are kept in sync.

In addition, file sync ensures that your Puppet code is deployed only when it is ready. These deployments ensure that your agents' code won't 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.

File sync works with Code Manager, so you typically won't need to do anything with file sync directly. If you want to know more about how file sync works, or you need to work with file sync directly for testing, this page provides additional information.

### File sync terms

There are a few terms that are helpful when you are working with file sync or tools that use file sync.

**Master of masters (or MoM)**

This is your "main" Puppet master. It typically serves as the Certificate Authority for all of your other masters, as well as for all of your agent nodes. In the context of file sync, it is also responsible for maintaining the canonical copy of all of your Puppet code and making it available to compile masters.

**Compile master**

These are secondary masters, generally added to your infrastructure as needed to scale up to the amount of load generated by your fleet of agents. They are typically set up behind a load balancer. In the context of file sync, these masters acquire the latest version of your Puppet code from the master of masters.

**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 may see this referred to simply as live-dir. This directory exists on all of your masters.

**Staging code directory**

This is the directory in which you should 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 all of your masters. This directory exists only on the master of masters; the compile masters 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 all of your masters and agents.

By default, file sync is disabled and the staging directory is not created on the MoM. 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 all masters. This creates the staging directory on the MoM, 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
compile masters. Normally, Code Manager triggers this commit automatically, but you can trigger a commit by hitting the file sync endpoint.

For example:

```
/opt/puppetlabs/puppet/bin/curl -s --request POST --header "Content-Type: application/json" --data "{"commit-all": true}" --cert ${cert} --key ${key} --cacert ${cacert} https://${fqdn}:8140/file-sync/v1/commit"
```

The above command is run from the MoM and contains the following variables:

- **fqdn**: Fully qualified domain name of the MoM.
- **cacert**: The Puppet CA’s certificate (/etc/puppetlabs/puppet/ssl/certs/ca.pem).
- **cert**: The ssl cert for the MoM (/etc/puppetlabs/ssl/certs/${fqdn}.pem).
- **key**: The private key for the MoM (/etc/puppetlabs/ssl/private_keys/${fqdn}.pem).

This command commits all of the changes in the staging directory. After the commit, when any compile masters 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 masters. (By default, compile masters 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 master of masters, 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 will create fresh repositories on each client and the storage server for the code it manages.

**Checking your deployments**

You can manually check information about file sync's deployments with curl commands that hit the *status* endpoint.

To manually check that your code has been successfully committed and deployed, you can hit a status endpoint on the MoM.

```
curl -k https://${fqdn}:8140/status/v1/services?level=debug
```

This returns output in JSON, so you can pipe it to `python -m json.tool` for better readability.

To check a list of file sync's clients, query the `file-sync-storage-service` section of the MoM by running a curl command.

This 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.

```
curl -k https://${fqdn}:8140/status/v1/services/file-sync-client-service?level=debug
```

If your commit has been deployed, it is listed in the status listing for `latest_commit`.

**Cautions**

There are a few things you should be aware of with file sync.

**Always use the staging directory**

Always make changes to your Puppet code in the staging directory. If you have edited code in the live code directory on the MoM or any compiled masters, *it will be overwritten* by file sync on the next commit.

The `enable-forceful-sync` parameter is set to *true* by default in PE. When *false*, file sync will not overwrite changes in the code directory, but instead logs errors in `/var/log/puppetlabs/puppetserver/puppetserver.log`. To set this parameter to *false*, add via 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. If you are using file sync, specify modules in the Puppetfile and use Code Manager to handle your syncs.

**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 directory. To make sure file sync can read and write what it needs to, ensure that these code directories are both owned by the `pe-puppet` user:

```
chown -R pe-puppet /etc/puppetlabs/code /etc/puppetlabs/code-staging
```
Environment isolation metadata

File sync generates .pp metadata files in both your live and staging code directories. These files provide environment isolation for your resource types, ensuring that each environment uses the correct version of the resource type. 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, see environment isolation.

Provisioning with Razor

Razor is a provisioning application that deploys bare-metal systems.

Policy-based provisioning lets you use characteristics of the hardware as well as user-provided data to make provisioning decisions. You can automatically discover bare-metal hardware, dynamically configure operating systems and hypervisors, and hand off nodes to Razor for workload configuration.

Automated provisioning makes Razor ideal for big installation jobs, like setting up a new selection of servers in a server farm. You can also use Razor to regularly wipe and re-provision test machines.
• **How Razor works** on page 624
  There are five key steps for provisioning nodes with Razor.
  
  • **Setting up a Razor environment** on page 626
  Razor relies on a PXE environment to boot the Razor microkernel. You must set up your PXE environment before you can successfully provision with Razor.
  
  • **Installing Razor** on page 628
  After you set up a Razor environment, you're ready to install Razor.
  
  • **Using the Razor client** on page 634
  There are three ways to communicate with the Razor server.
  
  • **Protecting existing nodes** on page 659
  In *brownfield environments* – those in which you already have machines installed that PXE boot against the Razor server – you must take extra precautions to protect existing nodes. Failure to adequately protect existing nodes can result in data loss.
  
  • **Provisioning a *nix node** on page 660
  Provisioning deploys and installs your chosen operating system to target nodes.
  
  • **Provisioning a Windows node** on page 669
  Provisioning deploys and installs your chosen operating system to target nodes.
  
  • **Provisioning with custom facts** on page 674
  You can use a microkernel extension to provision nodes based on hardware info or metadata that isn’t available by default in Facter.
  
  • **Working with Razor objects** on page 676
  Provisioning with Razor requires certain objects that define how nodes are provisioned.
  
  • **Using the Razor API** on page 690
  The Razor API is REST-based.
  
  • **Upgrading Razor** on page 704
  If you used Razor in a previous Puppet Enterprise environment, upgrade Razor to keep your Puppet Enterprise and Razor versions synched.
  
  • **Uninstalling Razor** on page 705
  If you're permanently done provisioning nodes, you can uninstall Razor.

### How Razor works

There are five key steps for provisioning nodes with Razor.

![Figure 2: Razor identifies a node](image)

**Figure 2: Razor identifies a node**

When a new node appears, Razor discovers its characteristics by booting it with the Razor microkernel and using Facter to inventory its facts.
The node is tagged based on its characteristics. Tags contain a match condition — a Boolean expression that has access to the node’s facts and determines whether the tag should be applied to the node or not.

Node tags are compared to tags in the policy table. The first policy with tags that match the node’s tags is applied to the node.

The policy governs how the node is installed.

Respositories
The content to install on a node. To create a repository, either import an install ISO or point at an existing package repository.

Tasks
Installation scripts such as kickstart files, prepare files and additional shell scripts. Predefined tasks are shipped with Razor, and custom tasks can easily be added without additional coding.

Brokers
Pre-installation scripts that install a config management agent on the node and enroll the node with the config management system. A predefined broker type for Puppet Enterprise is shipped with Razor.

Tags
Boolean expressions that use node facts and metadata. Tags are used to match nodes and policies.

Misc. Data
Makes it possible to generate hostnames for nodes, limit the number of nodes that can match that policy, and initialize a node’s metadata.

Figure 3: The node is tagged

Figure 4: The node tags match a Razor policy

Figure 5: Policies pull together all the provisioning elements
Figure 6: Razor provisions the node

The node is now installed. If you choose, you can hand off management of the node to Puppet Enterprise.

Razor system requirements

The Razor server and client are supported on these operating systems.

<table>
<thead>
<tr>
<th>Component</th>
<th>Supported OS</th>
</tr>
</thead>
</table>
| Server    | • RHEL 6 and 7  
            | • CentOS 6 and 7 |
| Client    | • Windows 2008 R2 Server  
            | • Windows 2012 R2 Server  
            | • Windows 2016 Server  
            | • ESXi 5.5  
            | • RHEL 6 and 7  
            | • CentOS 6 and 7  
            | • Ubuntu 14.04  
            | • Debian Wheezy |

Pre-requisites for machines provisioned with Razor

To successfully install an operating system on a machine using Razor, the machine must meet these specifications.

• Have at least 512MB RAM.
• Be supported by the operating system you're installing.
• Be able to successfully boot into the microkernel. The microkernel is based on CentOS 7, 64-bit only, and supports the x86-64 Intel architecture.
• Be able to successfully boot the iPXE firmware.

Setting up a Razor environment

Razor relies on a PXE environment to boot the Razor microkernel. You must set up your PXE environment before you can successfully provision with Razor.

There are two ways to enable PXE boot:

• (Recommended) Integrate Razor with the DHCP solution of the system being provisioned.
• Rely on UEFI to directly load the .ipxe file that’s required to boot the Razor microkernel.

Because it’s difficult to guarantee that all of your hardware is UEFI-enabled, configuring DHCP is the preferred method for setting up a Razor environment.
Set up a Razor environment

Set up a PXE environment using the DHCP and TFTP service of your choice.

**Before you begin**
You must have Puppet Enterprise installed.

This workflow describes a sample setup using dnsmasq. Dnsmasq is not intended for production environments; however, you can use a similar workflow to set up DHCP and TFTP with more robust solutions.

**Important:** Set up and test PXE in a completely isolated test environment. Running a second DHCP server on your company’s network could bring down the network or replace a server with a fresh installation. See Protecting existing nodes for strategies on avoiding data loss.

**Install and configure dnsmasq DHCP-TFTP service**
Install dnsmasq to manage communication between nodes and the Razor server. When a node boots, dnsmasq forwards the booted node to the Razor service.

1. Use YUM to install dnsmasq: `yum install dnsmasq`
2. If it doesn't already exist, create the directory `/var/lib/tftpboot`.
3. Change the permissions for the TFTP boot directory: `chmod 655 /var/lib/tftpboot`.

**Temporarily disable SELinux**
You must temporarily disable SELinux in order to enable PXE boot. Alternatively, you could craft an enforcement rule for SELinux that enables PXE boot but doesn't completely disable SELinux.

1. In the file `/etc/sysconfig/selinux`, set `SELINUX=disabled`.
2. Restart the computer.

**Edit dnsmasq.conf to enable DHCP**
You must specify a DHCP range to enable communication with your DHCP server.

**Tip:** For more complex setups, it might be helpful to indicate a range name, for example: `dhcp-range=rangel,10.0.1.50,10.0.1.120,24h dhcp-range=range2,10.0.1.121,10.0.1.222,48h`.

Edit `/etc/dnsmasq.conf` to specify a DHCP range.

For example, for an IP range from 10.0.1.50 - 10.0.1.120 with a 24-hour lease, your file resembles:

```conf
# Uncomment this to enable the integrated DHCP server, you need
# to supply the range of addresses available for lease and optionally
# a lease time. If you have more than one network, you will need to
# repeat this for each network on which you want to supply DHCP
# service.
dhcp-range=10.0.1.50,10.0.1.120,24h
```

**Edit the dnsmasq configuration file to enable PXE boot**
Use dnsmasq to enable PXE booting on nodes.

1. At the bottom of the `/etc/dnsmasq.conf` file, add: `conf-dir=/etc/dnsmasq.d`
2. Write and exit the file.
3. Create the file `/etc/dnsmasq.d/razor` and add configuration information.

   For example, for dnsmasq 2.45:

   ```
   # iPXE sets option 175, mark it for network IPXEBOOT
   dhcp-match=IPXEBOOT,175
   dhcp-boot=net:IPXEBOOT,bootstrap.ipxe
   dhcp-boot=undionly-20140116.kpxe
   # TFTP setup
   enable-tftp
   tftp-root=/var/lib/tftpboot
   ```

4. Enable dnsmasq on boot: `chkconfig dnsmasq on`

5. Start the dnsmasq service: `service dnsmasq start`

### Installing Razor

After you set up a Razor environment, you're ready to install Razor.

#### Install Razor

At a minimum, to install Razor, you must install the Razor server. You can also install the client to make interacting with the server easier, and enable authentication security to control permissions.

**Tip:** Use variables to avoid repeatedly replacing placeholder text. For installation, we recommend declaring a server name and the port to use for Razor with these commands:

```
export RAZOR_HOSTNAME=<server name>
export HTTP_PORT=8150
export HTTPS_PORT=8151
```

The installation tasks on this page use `$<RAZOR_HOSTNAME>`, `$HTTP_PORT`, and `$HTTPS_PORT` to represent these variables.

#### Install the Razor server

Installing Razor involves classifying a node with the `pe_razor` module.

When Puppet Enterprise applies this classification, the software downloads automatically and installs a Razor server and a PostgreSQL database. The download can take several minutes.

Because the Razor software is stored online, you need an internet connection to install it.

**Important:** Don't install the Razor server on your master.

1. In the console, create a classification node group for the server.
2. Click the server node group, click Configuration, then in the Add new class field, enter `pe_razor` and click Add class.
3. Commit changes.
4. On the server, run: `puppet agent -t`

#### Related information

- [Create classification node groups](#) on page 367
- Create classification node groups to assign classification data to nodes.
- [Running Puppet on nodes](#) on page 364
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.

**Install the Razor server while you’re offline**

Install Razor using your own remote directory containing a Puppet Enterprise tarball and the Razor microkernel.

**Before you begin**

Copy the **Razor microkernel** and the **Puppet Enterprise tarball** appropriate for your installation to your own FTP site or to the Razor server.

If you save the files locally, when you specify directory paths, use three forward slashes (`file:///`). If you save the files on the Razor server, use two forward slashes (`file://`).

The example parameter values in this task use this directory structure:

```plaintext
[root@razor ~]# tree /tmp/razor_files/
/tmp/razor_files/
   ### 2017.2.3/
     #   ### puppet-enterprise-2017.2.3-el-7-x86_64.tar.gz
     ### microkernel-008.tar
```

1. In the console, create a classification node group for the server.
2. Click the server node group, click **Configuration**, then in the **Add new class** field, enter `pe_razor` and click **Add class**.
3. In the **pe_razor** class, specify the URL for your tarball, and then click **Add parameter**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
| `pe_tarball_base_url` | Full path to a directory that contains `<PE_VERSION>`/<PE_INSTALLER>.tar.gz, for example `file:///tmp/razor_files`.
The complete URL is automatically constructed by appending the PE build and file name to the base directory.  |

4. In the **pe_razor** class, specify the URL for the microkernel, and then click **Add parameter**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>microkernel_url</code></td>
<td>Full path to the microkernel-&lt;VERSION&gt;.tar, for example <code>file:///tmp/razor_files/microkernel-008.tar</code>.</td>
</tr>
</tbody>
</table>

5. Commit changes.
6. On the server, run `puppet agent -t`
7. (Optional) Install the Razor client.
   a) From a web-enabled machine, fetch these gems:

   ```bash
   /opt/puppetlabs/puppet/bin/gem fetch colored --version 1.2
   /opt/puppetlabs/puppet/bin/gem fetch command_line_reporter --version 3.3.6
   /opt/puppetlabs/puppet/bin/gem fetch mime-types --version 1.25.1
   /opt/puppetlabs/puppet/bin/gem fetch multi_json --version 1.12.1
   /opt/puppetlabs/puppet/bin/gem fetch razor-client --version 1.9.4
   /opt/puppetlabs/puppet/bin/gem fetch faraday --version 0.15.4
   ```

   b) Copy the downloaded gems to a directory on the Razor server.

   c) Install the gems on the Razor server:

   ```bash
   /opt/puppetlabs/puppet/bin/gem install -f --local * .gem
   ```

**Related information**

Create classification node groups on page 367
Create classification node groups to assign classification data to nodes.

Running Puppet on nodes on page 364
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.

**Change the default Razor port**

The default ports for Razor are port 8150 for HTTP communication between the server and nodes, and port 8151 for HTTPS, used for accessing the public API. You can optionally change the default ports if they’re occupied by another service, out of range, or blocked by a firewall.

1. In the console, select the Classification, then click the node group that contains the **pe_razor** module.

2. On the Configuration tab, under **pe_razor** in the Parameters box, select whether you want to change **server_http_port** or **serverHttps_port**.

3. In the Value field, enter the port number to use, click Add parameter, and then commit changes.

**Parameters for the **pe_razor** class**

Parameters for the **pe_razor** class enable customization of your Razor installation. You can review configuration settings that are currently being used by Razor with the **razor config** command.

The **pe_razor** class has these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>api_config_blacklist</td>
<td>['facts.blacklist', 'database_url']</td>
<td>Properties that Razor hides from query results. You can add additional properties to protect sensitive data.</td>
</tr>
<tr>
<td>auth_config</td>
<td>/etc/puppetlabs/razor-server/shiro.ini</td>
<td>Path to the authentication configuration file.</td>
</tr>
<tr>
<td>auth_enabled</td>
<td>false</td>
<td>true to enable authentication for requests to /api endpoints, or false.</td>
</tr>
<tr>
<td>broker_path</td>
<td>/etc/puppetlabs/razor-server/brokers:brokers</td>
<td>Colon-separated list of directories containing broker types.</td>
</tr>
<tr>
<td>checkin_interval</td>
<td>15</td>
<td>Interval, in seconds, at which the microkernel checks in with the Razor server.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dbpassword</td>
<td>razor</td>
<td>Password to the Razor database.</td>
</tr>
<tr>
<td>facts_blacklist</td>
<td>['domain', 'filesystems', 'fqdn', 'hostname', 'id', '/kernel.<em>/', 'memoryfree', 'memorysize', 'memorytotal', '/operatingsystem.</em>/', 'osfamily', 'path', 'ps', 'rubysitedir', 'rubyversion', 'selinux', 'sshdsakey', '/sshfp_[dr]sa/', 'sshrsakey', '/swap.<em>/', 'timezone', '/uptime.</em>/']</td>
<td>Facts that Razor ignores. Each entry may be a string or a regexp enclosed in /../ where any fact that matches the regexp is dropped.</td>
</tr>
<tr>
<td>facts_match_on</td>
<td>[]</td>
<td>Array of values used to match nodes from within the microkernel to nodes in the Razor database. By default, this parameter excludes / ^macaddress.##42;/ (regex), serialnumber, and uuid, which are already used.</td>
</tr>
<tr>
<td>hook_execution_path</td>
<td>/opt/puppetlabs/puppet/bin</td>
<td>Colon-separated list of paths that Razor searches in order when running hooks, prior to using the default execution path.</td>
</tr>
<tr>
<td>hook_path</td>
<td>/etc/puppetlabs/razor-server/hooks:hooks</td>
<td>Colon-separated list of directories containing hook types.</td>
</tr>
<tr>
<td>match_nodes_on</td>
<td>['mac']</td>
<td>Array of values used to match nodes when a node PXE boots. Values can include mac, serial, asset, or uuid.</td>
</tr>
<tr>
<td>microkernel_debug_level</td>
<td>quiet</td>
<td>Sets the logging level for the microkernel. Valid values are quiet or debug.</td>
</tr>
<tr>
<td>microkernel_extension_zip</td>
<td>/etc/puppetlabs/razor-server/mk-extension.zip</td>
<td>Zip file that specifies custom facts or other code that is unpacked by the microkernel prior to checkin.</td>
</tr>
<tr>
<td>microkernel_kernel_args</td>
<td>&quot;</td>
<td>Additional command-line arguments that are supplied to the microkernel during boot.</td>
</tr>
<tr>
<td>microkernel_url</td>
<td><a href="https://pm.puppetlabs.com/puppet-enterprise-razor-microkernel-$">https://pm.puppetlabs.com/puppet-enterprise-razor-microkernel-$</a></td>
<td>Location of the Razor microkernel used to install Razor offline.</td>
</tr>
<tr>
<td>pe_tarball_base_url</td>
<td><a href="https://pm.puppetlabs.com/puppet-enterprise">https://pm.puppetlabs.com/puppet-enterprise</a></td>
<td>Location of the Puppet Enterprise tarball used to install Razor offline.</td>
</tr>
<tr>
<td>protect_new_nodes</td>
<td>false</td>
<td>true to make new machines ineligible for provisioning, or false.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>repo_store_root</td>
<td>/opt/puppetlabs/server/data/razor-server/repo</td>
<td>Directory where repository contents are downloaded and served.</td>
</tr>
<tr>
<td>secure_api</td>
<td>true</td>
<td>true to require HTTPS/SSL communication with /api endpoints.</td>
</tr>
<tr>
<td>server_http_port</td>
<td>8150</td>
<td>Port that nodes use to communicate with the server over HTTP. Only URLs starting with /svc need to be available on this port.</td>
</tr>
<tr>
<td>server_https_port</td>
<td>8151</td>
<td>Port that the client uses to communicate with the server's public API over HTTPS. Only URLs starting with /api need to be available on this port.</td>
</tr>
<tr>
<td>task_path</td>
<td>/etc/puppetlabs/razor-server/tasks:tasks</td>
<td>Colon-separated list of directories containing tasks.</td>
</tr>
</tbody>
</table>

**Verify the Razor server**

Use a test command to verify that the Razor server is correctly installed. The output JSON file test.out contains a list of available Razor commands.

Test the Razor configuration:

```
wget https://$<RAZOR_HOSTNAME>:$<HTTPS_PORT>/api -O test.out
```

**Install the Razor client**

The Razor client is installed as a Ruby gem, razor-client. The process for installing the client differs by platform.

**Install the Razor client on *nix systems**

Make interacting with the Razor server from *nix systems easier by installing the Razor client.

1. Install the client: `gem install razor-client`
   
   **Tip:** If you're installing the Razor client on your master, you must specify the location of the gem: `/opt/puppetlabs/puppet/bin/gem install razor-client`

2. Point the Razor client to the server: `razor -u https://$<RAZOR_HOSTNAME>:$<HTTPS_PORT>/api`
   
   An error displays if the client isn’t installed or can’t connect to the server.

3. (Optional) If you receive a warning message about JSON, you can optionally disable it: `gem install json_pure`

**Install the Razor client on Windows systems**

Make interacting with the Razor server from Windows systems easier by installing the Razor client.

1. From a Puppet command prompt (Start > Start Command Prompt with Puppet > Run as administrator), install the client: `gem install razor-client`

   
   **Tip:** Alternatively, you can set the variable through User Accounts in the Control Panel.
Enable authentication security

Enable authentication security to control what tasks users can perform. For example, you might limit certain users to read permissions to avoid accidental overwrite of nodes.

Two methods are required to secure your Razor server:

• Authentication security using Shiro – disabled by default
• Protocol security using HTTPS and TLS/SSL – enabled by default if you’re using Puppet Enterprise and you install Razor on a managed node

1. In the console, in the pe_razor class, specify Shiro authentication parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_enabled</td>
<td>true</td>
</tr>
<tr>
<td>auth_configured</td>
<td>Location of the Shiro file, /etc/puppetlabs/razor-server/shiro.ini</td>
</tr>
</tbody>
</table>

2. Specify users, passwords, and roles in your shiro.ini file, located at /etc/puppetlabs/razor-server/shiro.ini.

For example, this INI file specifies two users: razor is an admin who can perform all functions, and other is a user who can perform only read operations, such as viewing collections.

```
[main]
sha256Matcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher
iniRealm.credentialsMatcher = $sha256Matcher

[users]
# define users known to shiro, using the format:
# <username> = <sha256 password hash>, <role>[, <role>...]
razor = 9b4f1d0e11dcc029c3493d945e44ee077b68978466c0aab6d1ce453aac5f0384, admin
other = d74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a725000feb82e8f1, user

[roles]
# define roles and their associated permissions
admin = *
user = query:*
```

3. Restart the Razor service: `service pe-razor-server restart`

Permissions for Razor

These are the available permissions for Razor.

**Note:** Commands have varying access control patterns. Use `razor <COMMAND_NAME> --help` to view required permissions for each command.

<table>
<thead>
<tr>
<th>Task</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query all collections</td>
<td>query:*</td>
</tr>
<tr>
<td>Query the node collection</td>
<td>query:nodes</td>
</tr>
<tr>
<td>Query a specific node</td>
<td>query:nodes:&lt;NODE_NAME&gt;</td>
</tr>
<tr>
<td>Run all commands</td>
<td>commands:*</td>
</tr>
<tr>
<td>Create policies</td>
<td>commands:create-policy:*</td>
</tr>
<tr>
<td>Create policies starting with a specific name</td>
<td>commands:create-policy:&lt;NAME&gt;*</td>
</tr>
</tbody>
</table>
Verify Razor versions
Verify your installation by checking the version of the Razor server and client. This information can also be useful for troubleshooting.

On the client or server, verify versions: `razor --version` or `razor -v`

Using the Razor client
There are three ways to communicate with the Razor server.

- API calls in JSON sent directly to the server
- JSON arguments sent from the Razor client to the server
- Razor client commands

Client commands are the easiest way to interact with the server. Client commands begin with `razor`, followed by the name of the action, like `razor create-repo` or `razor move-policy`. One or more arguments follow the command.

You can access help for each command by entering the command with the `--help` flag, for example `razor add-policy-tag --help`.

Using positional arguments with Razor client commands
Most Razor client commands allow positional arguments, which means that you don’t have to explicitly enter the name of the argument, like `--name`. Instead, you can provide the values for each argument in a specific order.

For example, the `delete-policy` command includes only one argument, `--name`. To delete a policy named `sprocket`, you can enter the command with the argument name and value, or with a positional argument:

- command with argument name and value — `razor delete-policy --name sprocket`
- command with positional argument — `razor delete-policy sprocket`

If a command includes multiple options, you can supply from zero to all available positional arguments. For example, the `add_policy_tag` command has three positional arguments: `name`, `tag`, and `rule`. You can provide no positional arguments, `name` only, `name` and `tag` only, or all three arguments.

Because not all arguments are available as positional arguments, you might need to use a combination of positional arguments and name-value pairs. For example, the `create-hook` command has two positional arguments, `name` and `hook-type`, which you might use along with a `--configuration` value, like:

```
razor create-hook name_of_hook hook_type --configuration someconfig=value
```

You can switch between positional and non-positional arguments, but you must maintain the expected order for positional arguments. For example:

```
razor command positional-arg1 --non-positional value positional-arg2 --non-positional2
```

Positional arguments for client commands
These are the Razor client commands, with available positional arguments listed in accepted order.

<table>
<thead>
<tr>
<th>Command</th>
<th>Positional arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>add-policy-tag</td>
<td>name, tag, rule</td>
</tr>
<tr>
<td>create-broker</td>
<td>name, broker-type</td>
</tr>
<tr>
<td>create-hook</td>
<td>name, hook-type</td>
</tr>
<tr>
<td>create-policy</td>
<td>name</td>
</tr>
<tr>
<td>Command</td>
<td>Positional arguments</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>create-repo</td>
<td>name</td>
</tr>
<tr>
<td>create-tag</td>
<td>name, rule</td>
</tr>
<tr>
<td>create-task</td>
<td>name</td>
</tr>
<tr>
<td>delete-broker</td>
<td>name</td>
</tr>
<tr>
<td>delete-hook</td>
<td>name</td>
</tr>
<tr>
<td>delete-node</td>
<td>name</td>
</tr>
<tr>
<td>delete-policy</td>
<td>name</td>
</tr>
<tr>
<td>delete-repo</td>
<td>name</td>
</tr>
<tr>
<td>delete-tag</td>
<td>name</td>
</tr>
<tr>
<td>disable-policy</td>
<td>name</td>
</tr>
<tr>
<td>enable-policy</td>
<td>name</td>
</tr>
<tr>
<td>modify-node-metadata</td>
<td>name</td>
</tr>
<tr>
<td>modify-policy-max-count</td>
<td>name, max_count</td>
</tr>
<tr>
<td>move-policy</td>
<td>name</td>
</tr>
<tr>
<td>reboot-node</td>
<td>name</td>
</tr>
<tr>
<td>register-node</td>
<td>(none)</td>
</tr>
<tr>
<td>reinstall-node</td>
<td>name</td>
</tr>
<tr>
<td>remove-node-metadata</td>
<td>node, key</td>
</tr>
<tr>
<td>remove-policy-tag</td>
<td>name, tag</td>
</tr>
<tr>
<td>run-hook</td>
<td>name</td>
</tr>
<tr>
<td>set-node-desired-power-state</td>
<td>name, to</td>
</tr>
<tr>
<td>set-node-hw-info</td>
<td>node</td>
</tr>
<tr>
<td>set-node-ipmi-credentials</td>
<td>name</td>
</tr>
<tr>
<td>update-broker-configuration</td>
<td>broker, key, value</td>
</tr>
<tr>
<td>update-hook-configuration</td>
<td>hook, key, value</td>
</tr>
<tr>
<td>update-node-metadata</td>
<td>node, key, value</td>
</tr>
<tr>
<td>update-policy-repo</td>
<td>policy, repo</td>
</tr>
<tr>
<td>update-policy-task</td>
<td>policy, task</td>
</tr>
<tr>
<td>update-policy-broker</td>
<td>policy, broker</td>
</tr>
<tr>
<td>update-policy-node-metadata</td>
<td>policy, key, value</td>
</tr>
<tr>
<td>update-repo-task</td>
<td>repo, task</td>
</tr>
<tr>
<td>update-tag-rule</td>
<td>name, rule</td>
</tr>
</tbody>
</table>
Razor client commands

Use client commands to interact with Razor and provision bare metal nodes.

Configuration commands
Razor configuration is pulled from a configuration file controlled by Puppet Enterprise. You can change configuration values in the console with class parameters of the pe_razor module.

Note: In order for configuration changes to take effect, you must restart the Razor service: `service pe-razor-server restart`.

config
The config command displays details about your Razor configuration.

Note: Properties specified in the api_config_blacklist aren’t returned by the config command.

Sample command
To view details about your Razor configuration:

    razor config

Node commands
Nodes are created either through the node boot endpoint, when the node initiates its first web request to the Razor server, or through the register-node command.

Both methods of creating a node create a stub in the Razor database. When the node boots into the microkernel, it gathers facts and reports them to the node checkin endpoint, linking new facts to the stub. At that point, the node is fully registered in the Razor system and goes through the binding process.

register-node
The register-node command enables you to identify a node before it’s discovered. This can be useful to apply metadata before the node boots up, or to indicate to Razor that it shouldn’t provision – or reprovision – a node that’s already installed.

Command attributes
Attributes | Required | Positional arguments | Description
--- | --- | --- | ---
hw_info | # |  | Hardware information about the node. You must include enough information that the node can be identified by hardware information sent by the firmware when the node boots. This usually includes the MAC addresses of all network interfaces. This attribute can include some or all of these entries:
• netN — The MAC address of each network interface. The order of addresses isn’t significant.
• serial — DMI serial number of the node.
• asset — DMI asset number of the node.
• uuid — DMI universally unique identifier of the node.

installed | # |  | true to indicate that the node shouldn’t be provisioned — or reprovisioned — by Razor, or false.

Sample command
To register an installed machine before it’s booted, and prevent reprovisioning:

```bash
razor register-node --hw-info net0=78:31:c1:be:c8:00 \ 
  --hw-info net1=72:00:01:f2:13:f0 \ 
  --hw-info net2=72:00:01:f2:13:f1 \ 
  --hw-info serial=xxxxxxxxxxx \ 
  --hw-info asset=Asset-1234567890 \ 
  --hw-info uuid="Not Settable" \ 
  --installed
```

`set-node-hw-info`
The `set-node-hw-info` command sets or updates the hardware info for a specified node. This is useful when a node’s hardware changes, such as when you replace a network card.

Command attributes

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>#</td>
<td>1</td>
<td>Name of the node to update.</td>
</tr>
<tr>
<td>hw_info</td>
<td>#</td>
<td></td>
<td>Hardware information about the node. This attribute can include some or all of these entries:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• netN — The MAC address of each network interface. The order of addresses isn't significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• serial — DMI serial number of the node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• asset — DMI asset number of the node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• uuid — DMI universally unique identifier of the node.</td>
</tr>
</tbody>
</table>

**Sample command**

To update node172 with new hardware information:

```
razor set-node-hw-info --node node172  
 --hw-info net0=78:31:c1:be:c8:00  
 --hw-info net1=72:00:01:f2:13:f0  
 --hw-info net2=72:00:01:f2:13:f1  
 --hw-info serial=xxxxxxxxxxx  
 --hw-info asset=Asset-1234567890  
 --hw-info uuid="Not Settable"
```

**delete-node**

The `delete-node` command deletes a specified node from the Razor database. The `delete-node` command is similar to `reinstall-node`, except that `delete-node` doesn’t retain log information or the node number.

**Note:** If the deleted node boots again at some point, Razor automatically recreates the node as if it were the first time it contacted the Razor server.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the node to delete.</td>
</tr>
</tbody>
</table>

**Sample command**

To delete node17:

```
razor delete-node --name node17
```

**reinstall-node**

The `reinstall-node` command clears a node’s `installed` flag and – by default – removes its association with policies. You can use the `same-policy` attribute to retain the assigned policy.
After restart, the node boots into the microkernel, goes through discovery and tag matching, and can bind to another policy for reinstallation. This command doesn’t change the node’s metadata or facts.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the node to reinstall.</td>
</tr>
<tr>
<td>same-policy</td>
<td></td>
<td></td>
<td>true to retain the same policy for reinstallation, or false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If omitted, the node can bind to a different policy when reinstalled.</td>
</tr>
</tbody>
</table>

**Sample commands**

To reinstall node17:

```
razor reinstall-node --name node17
```

To reinstall node17 and retain its assigned policy:

```
razor reinstall-node --name node17 --same-policy
```

**IPMI commands**

IPMI commands are node commands based on the Intelligent Platform Management Interface.

**Note:** You must install `ipmitool` on the Razor server before using IPMI commands. To install the tool, run `yum install ipmitool -y`.

**set-node-ipmi-credentials**

The `set-node-ipmi-credentials` sets or updates the host name, user name, or password for connecting with a BMC/LOM/IPMI LAN or LANplus service.

The command works only with remote IPMI targets, not local targets.

After IPMI credentials have been set up for a node, you can use the `reboot-node` and `set-node-desired-power-state` commands.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the node to update.</td>
</tr>
<tr>
<td>ipmi_hostname</td>
<td></td>
<td></td>
<td>IPMI host name or IP address of the BMC of the host.</td>
</tr>
<tr>
<td>ipmi_username</td>
<td></td>
<td></td>
<td>IPMA LANplus username, if any, for this BMC.</td>
</tr>
<tr>
<td>ipmi_password</td>
<td></td>
<td></td>
<td>IPMI LANplus password, if any, for this BMC.</td>
</tr>
</tbody>
</table>

**Sample command**

© 2020 Puppet, Inc.
To set IPMI credentials for node17:

```bash
terminal
razor set-node-ipmi-credentials --name node17 \
    --ipmi-hostname bmc17.example.com \
    --ipmi-username null \
    --ipmi-password sekretswirr1
```

**reboot-node**

The `reboot-node` command triggers a hard power cycle on a specified node using IPMI credentials.

**Note:** You must specify IPMI credentials before using the `reboot-node` command.

If an execution slot isn’t available on the target node, the `reboot-node` command is queued and runs as soon as a slot is available. There are no limits on how many commands you can queue up, how frequently a node can be rebooted, or how long a command can stay in the queue. If you restart your Razor server before queued commands are executed, they remain in the queue and run after the server restarts.

The `reboot-node` command is not integrated with IPMI power state monitoring, so you can’t see power transitions in the record or when polling the node object.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the node to reboot.</td>
</tr>
</tbody>
</table>

**Sample command**

To queue a node reboot:

```bash
terminal
razor reboot-node --name node1
```

**set-node-desired-power-state**

The `set-node-desired-power-state` command specifies whether you want a node to remain powered off or on. By default, Razor checks node power state every few minutes in the background. If it detects a node in a non-desired state, Razor issues an IPMI command directing the node to its desired state.

**Note:** You must specify IPMI credentials before using the `set-node-desired-power-state` command.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the node to power off or on.</td>
</tr>
<tr>
<td>to</td>
<td></td>
<td>2</td>
<td>Desired power state: on, off, or null, meaning that Razor shouldn’t enforce a power state.</td>
</tr>
</tbody>
</table>

**Sample command**

To power on node1234 and keep it on:

```bash
terminal
razor set-node-desired-power-state --name node1234 \
    --to on
```
**Node metadata commands**

Node metadata commands enable you to update, modify, or remove metadata from nodes.

**update-node-metadata**

The `update-node-metadata` command sets or updates a single metadata key.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>#</td>
<td>1</td>
<td>Name of the node for which to modify metadata.</td>
</tr>
<tr>
<td>key</td>
<td>#</td>
<td>2</td>
<td>Name of the key to modify.</td>
</tr>
<tr>
<td>value</td>
<td>#</td>
<td>3</td>
<td>New value for the key.</td>
</tr>
<tr>
<td>no_replace</td>
<td></td>
<td></td>
<td><code>true</code> to cancel the update operation if the specified key is already present, or <code>false</code>.</td>
</tr>
</tbody>
</table>

**Sample command**

To update the `my_key` key for a node:

```
razor update-node-metadata --node node1 --key my_key --value twelve
```

**modify-node-metadata**

The `modify-node-metadata` command sets, updates, or clears metadata key-value pairs.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>#</td>
<td>1</td>
<td>Name of the node for which to modify metadata.</td>
</tr>
<tr>
<td>update</td>
<td>*</td>
<td></td>
<td>Key and value pair to update.</td>
</tr>
<tr>
<td>remove</td>
<td>*</td>
<td></td>
<td>Key and value pair to remove.</td>
</tr>
<tr>
<td>clear</td>
<td>*</td>
<td></td>
<td><code>true</code> to clear all metadata, or <code>false</code>.</td>
</tr>
<tr>
<td>no_replace</td>
<td></td>
<td></td>
<td><code>true</code> to cancel the update operation if the specified key is already present, or <code>false</code>.</td>
</tr>
<tr>
<td>force</td>
<td></td>
<td></td>
<td><code>true</code> to bypass errors in a batch operation with no_replace, or false.</td>
</tr>
</tbody>
</table>

* You must specify one of three attributes: `update`, `remove`, or `clear`. 
Sample commands

To add values for key1 and key2 to a node, but not if they are already set, and to remove key3 and key4:

```
razor modify-node-metadata --node node1 --update key1=value1 \ 
   --update key2='[ "val1", "val2", "val3" ]' --remove key3 --remove key4 --noreplace
```

To remove all node metadata:

```
razor modify-node-metadata --node node1 --clear
```

remove-node-metadata

The remove-node-metadata command removes either a single metadata entry or all metadata entries on a node.

Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>#</td>
<td>1</td>
<td>Name of the node for which to modify metadata.</td>
</tr>
<tr>
<td>key</td>
<td>*</td>
<td>2</td>
<td>Name of the key to remove.</td>
</tr>
<tr>
<td>all</td>
<td>*</td>
<td></td>
<td>true to remove all metadata about the node.</td>
</tr>
</tbody>
</table>

* You must specify one of two attributes: key or all.

Sample commands

To remove a single key from a node:

```
razor remove-node-metadata --node node1 --key my_key
```

To remove all keys from a node:

```
razor remove-node-metadata --node node1 --all
```

Repository commands

Repository commands enable you to create and delete specified repositories from the Razor database and specify the task that installs the contents of a repository.

create-repo

The create-repo command creates a new repository. The repository can contain the content to install a node, or it can point to an existing online repository.

You can create three types of repositories:

- Those that reference content available on another server, for example, on a mirror you maintain (url).
- Those where unpacks ISOs for you and serves their contents (iso_url).
- Those where creates a stub directory that you can manually fill with content (no_content).

Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name for the new repository.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Required</td>
<td>Positional arguments</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>url</td>
<td>*</td>
<td></td>
<td>URL of a remote repository.</td>
</tr>
<tr>
<td>iso_url</td>
<td>*</td>
<td></td>
<td>URL of an ISO image to download and unpack in a new, local repository. You can use an HTTP or HTTPS URL, or a file URL. If you're using a file URL, manually place the ISO image on the server before you call the command. If you supply the <code>iso_url</code> attribute to create a repository, you can delete it from the server later using the <code>delete-repo</code> command.</td>
</tr>
<tr>
<td>no_content</td>
<td>*</td>
<td></td>
<td>Creates a stub directory in the repo store where you can manually extract an image. After this command finishes, you can log into your server and fill the repository directory with the content, for example by loopback-mounting the install media and copying it into the directory. The repository directory is specified by the <code>repo_store_root</code> class parameter of the <code>pe_razor</code> module. By default, the directory is <code>/opt/puppetlabs/server/data/razor-server/repo</code>. This attribute is usually necessary for Windows install media, because the library that Razor uses to unpack ISO images doesn't support Windows ISO images.</td>
</tr>
<tr>
<td>task</td>
<td>#</td>
<td></td>
<td>Name of the default task that installs nodes from this repository.</td>
</tr>
</tbody>
</table>
We recommend that the task match the operating system specified by the `url` or `iso_url`. You can override this parameter at the policy level.

**Tip:** You can use `razor tasks` to see which tasks are available on your server.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>We recommend that the task match the operating system specified by the <code>url</code> or <code>iso_url</code>. You can override this parameter at the policy level. <strong>Tip:</strong> You can use <code>razor tasks</code> to see which tasks are available on your server.</td>
</tr>
</tbody>
</table>

* You must specify one of the attributes `url`, `iso_url`, or `no_content`.

**Sample commands**

To create a repository from an ISO image, which is downloaded and unpacked by the server in the background:

```
razor create-repo --name fedora21 --iso-url http://example.com/Fedora-21-x86_64-DVD.iso --task fedora
```

To unpack an ISO image from a file on a server without uploading the file from the client:

```
razor create-repo --name fedora21 --iso-url file:///tmp/Fedora-21-x86_64-DVD.iso --task fedora
```

To point to a resource without downloading content to the server:

```
razor create-repo --name fedora21 --url http://mirrors.n-ix.net/fedora/linux/releases/21/Server/x86_64/os/ --task fedora
```

To create a stub directory that you can manually fill with content:

```
razor create-repo --name fedora21 --no-content --task Fedora
```

**update-repo-task**

The `update-repo-task` command specifies the task that installs the contents of the repository.

If a node is currently provisioning against the repository when you run this command, provisioning might fail.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repo</td>
<td>#</td>
<td>1</td>
<td>Name of an existing repository to update.</td>
</tr>
<tr>
<td>task</td>
<td>#</td>
<td>2</td>
<td>Name of the task to be used by the repository.</td>
</tr>
</tbody>
</table>

**Sample command**

```
To update the my_repo repository to the task other_task:

```
razor update-repo-task --repo my_repo --task other_task
```

**delete-repo**

The delete-repo command deletes a specified repository from the Razor database.

Before deleting a repository, remove any references to it in existing policies. This command fails if the repository is in use by an existing policy.

If you supplied the iso_url property when you created the repository, the folder is also deleted from the server. If you didn’t supply the iso_url property, content remains in the repository directory.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the repository to delete.</td>
</tr>
</tbody>
</table>

**Sample command**

To delete an obsolete repository:

```
razor delete-repo --name my_obsolete_repo
```

**Task commands**

Use task commands to create tasks in the Razor database.

**Important:** tasks differ from tasks, which let you run arbitrary scripts and commands using . See the orchestrator documentation for information about tasks.

**create-task**

The create-task command creates a task in the Razor database. This command is an alternative to manually placing task files in the task_path. If you anticipate needing to make changes to tasks, we recommend the disk-backed task approach.

Razor has a set of tasks for installing on supported operating systems. See the razor-server Github page for more information.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name for the new task.</td>
</tr>
<tr>
<td>os</td>
<td>#</td>
<td></td>
<td>Name of the OS installed by the task.</td>
</tr>
<tr>
<td>templates</td>
<td>#</td>
<td></td>
<td>Named templates used for task stages.</td>
</tr>
<tr>
<td>boot_seq</td>
<td></td>
<td></td>
<td>List of key-value pairs that describe templates in the order they’re installed. Use the optional default hash key to specify the default</td>
</tr>
</tbody>
</table>
Sample command

To define the RedHat task included with Razor:

```bash
razor create-task --name redhat-new --os "Red Hat Enterprise Linux" \
  --description "A basic installer for RHEL6" \ 
  --boot-seq 1=boot_install --boot_seq default=boot_local \ 
  --templates "boot_install=#!ipxe\necho Razor <%= task.label %> task \nboot_call\nnacho Installation node: <%= node_url %>\nnecho Installation \nrepo: <%= repo_url %>\nsleep 3\nkernel <%= repo_url("/isolinux/vmlinuz") \n%> <%= render_template("kernel_args").strip %> || goto error\n
initrd <%= repo_url("/isolinux/initrd.img") %> || goto error\nboot -- \nkey s --timeout 60 ERROR, hit 's' for the iPXE shell; reboot in 60 seconds \n\ninitrd || reboot\n" \
  --templates kernel_args="ks=<%= file_url("kickstart") %> network \nksdevice=bootif BOOTIF=01-$(netX/mac)" \
  --templates kickstart="#!/bin/bash\n# Kickstart for RHEL/CentOS 6\n\nsee: http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/s1-kickstart2-options.html\n\ninstall\nurl --url=<%= repo_url %>
cmdline
lang en_US.UTF-8
keyboard us
rootpw <%= node.root_password %>
network --hostname <%= node.hostname %>
firewall --enabled --ssh
authconfig --enableshadow --passalgo=sha512
--enablefingerprint
timezone --utc America/Los_Angeles\n# Avoid\nhaving 'rhgb quiet' on the boot line\nbootloader --location=mbr --append="crashkernel=auto"\n\n%packages --nobase
@core
%end

%post --log=/var/log/razor.log
echo Kickstart post
curl -s \n\n# Run \nrazor_postinstall.sh on next boot via rc.local\nif 

\n\n\n%end

############
" \
  --templates post_install="#!/bin/bash\n
exec >> /var/log/razor.log 2>&1

echo "Starting post_install"

# Wait for network to come up when using NetworkManager.\n\nif service NetworkManager status >/dev/null 2>&1 && type -P nm-online; then\n  nm-online -q --timeout=10 \n  nm-online -q --timeout=30 \n  exit 1\nfi\n\nnf
bash /root/razor_postinstall.sh \n  /etc/rc.d/rc.local\n  chmod a+x /etc/rc.d/rc.local\n  nf\nnf
bash /root/razor_postinstall.sh \n  /etc/rc.d/rc.local\n  chmod a+x /root/razor_postinstall.sh \n  /etc/rc.d/rc.local\n  nm\n  curl -s \n  stage_done_url("kickstart") \n  %end

Tag commands

Tag commands enable you to create new tags, set the rules used to apply the tag to nodes, change the rule of a specified tag, or delete a specified tag.

create-tag

The create-tag command creates a new tag and sets the rules used to apply the tag to nodes.

Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>template to use when no other template applies.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Required</td>
<td>Positional arguments</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name for the new tag.</td>
</tr>
<tr>
<td>rule</td>
<td>#</td>
<td>2</td>
<td>Case-sensitive match expression that applies the tag to a node if the expression evaluates as true.</td>
</tr>
</tbody>
</table>

### Sample command
To create a tag that's applied to nodes with two processors:

```
razor create-tag --name small --rule "["="", ["fact", "processorcount"], "2"]"
```

### update-tag-rule
The **update-tag-rule** changes the rule of a specified tag.

After updating a tag, Razor reevaluates the tag against all nodes. With the **--force** flag, the tag is reevaluated even if it's used by policies.

#### Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of an existing tag to update.</td>
</tr>
<tr>
<td>rule</td>
<td>#</td>
<td>2</td>
<td>Case-sensitive match expression that applies the tag to a node if the expression evaluates as true.</td>
</tr>
<tr>
<td>force</td>
<td></td>
<td></td>
<td>Reevaluates tags used by an existing policy.</td>
</tr>
</tbody>
</table>

### Sample command
To update a tag rule and reevaluate nodes with the tag, even if the tag is used by a policy:

```
razor update-tag-rule --name small --force \n    --rule '["\<\="", ["fact", "processorcount"], "2"]'
```

### delete-tag
The **delete-tag** command deletes a specified tag.

With the **--force** flag, the tag is deleted and removed from any policies that use it.

#### Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of an existing tag to delete.</td>
</tr>
</tbody>
</table>
Attributes | Required | Positional arguments | Description
--- | --- | --- | ---
force |  |  | Deletes tags used by an existing policy.

Sample commands
To delete an unused tag:

```
razor delete-tag --name my_obsolete_tag
```

To delete a tag whether or not it is used:

```
razor delete-tag --name my_obsolete_tag --force
```

Related information
Tags on page 680
A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag's facts results in true. Tag matching is case sensitive.

Tags API on page 699
These commands enable you to create a tag, delete a tag, or change the rule for a tag.

Policy commands
Policies govern how nodes are provisioned.
Razor maintains an ordered table of policies. When a node boots, Razor traverses the table to find the first eligible policy for that node. A policy might be ineligible for binding to a node if the node doesn't contain all of the tags on the policy, if the policy is disabled, or if the policy has reached its maximum for the number of allowed nodes.

When you list the policies collection, the list is in the order that Razor checks policies against nodes.

create-policy
The create-policy command creates a new policy that determines how nodes are provisioned. Razor maintains an ordered table of policies, and applies the first policy that matches a node.

Tip: You can create policies in a JSON file to make saving and modifying them easier. To apply a policy with a JSON file, use razor create-policy --json <FILENAME>.json.

Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name for the new policy.</td>
</tr>
<tr>
<td>hostname</td>
<td>#</td>
<td></td>
<td>Pattern for host names of nodes bound to this policy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The ${id} references each node's internal ID number.</td>
</tr>
<tr>
<td>root_password</td>
<td>#</td>
<td></td>
<td>Root password for the new system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Valid values depend on how the task renders passwords.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Required</td>
<td>Positional arguments</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>enabled</td>
<td></td>
<td></td>
<td>true if the policy is enabled upon creation, or false. If omitted, the policy is enabled.</td>
</tr>
<tr>
<td>max_count</td>
<td></td>
<td></td>
<td>Number indicating how many nodes can bind to the policy. If omitted, the policy can bind to an unlimited number of nodes.</td>
</tr>
<tr>
<td>before</td>
<td></td>
<td></td>
<td>Name of the policy that this policy is placed before in the policy list. This attribute can't be used with after.</td>
</tr>
<tr>
<td>after</td>
<td></td>
<td></td>
<td>Name of the policy that this policy is placed after in the policy list. This attribute can't be used with before.</td>
</tr>
<tr>
<td>tags</td>
<td></td>
<td></td>
<td>Names of tags that a node must match to qualify for the policy. A node must have all tags specified for the policy to apply.</td>
</tr>
<tr>
<td>repo</td>
<td>#</td>
<td></td>
<td>Name of the repository that contains the operating system installed by this policy.</td>
</tr>
<tr>
<td>broker</td>
<td>#</td>
<td></td>
<td>Name of the broker to use after provisioning to hand off the node to a management system. If you don't want to hand off management, use noop.</td>
</tr>
<tr>
<td>task</td>
<td></td>
<td></td>
<td>Name of the task to install nodes that bind to this policy.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Required</td>
<td>Positional arguments</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>node_metadata</td>
<td></td>
<td></td>
<td>Metadata to apply to the node when this policy is bound. Existing metadata is not overwritten. To install Windows on non-English systems, specify the <code>win_language</code> using the culture code for the appropriate Microsoft language pack. For example, --node-metadata win_language=es-ES</td>
</tr>
</tbody>
</table>

**Sample command**

To create a policy that installs CentOS 6.4:

```bash
razor create-policy --name centos-for-small \  
--repo centos-6.4 --task centos --broker noop \  
--enabled --hostname "host${id}.example.com" \  
--root-password secret --max-count 20 \  
--before "other policy" --tag small --node-metadata key=value
```

**move-policy**

The `move-policy` command lets you change the order in which Razor considers policies for matching against nodes.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to move.</td>
</tr>
<tr>
<td>before</td>
<td>*</td>
<td></td>
<td>Name of the policy that this policy is placed before in the policy list.</td>
</tr>
<tr>
<td>after</td>
<td>*</td>
<td></td>
<td>Name of the policy that this policy is placed after in the policy list.</td>
</tr>
</tbody>
</table>

* You must specify one of the two ordering attributes, `before` or `after`.

**Sample commands**
To move a policy before another policy:

```bash
razor move-policy --name policy --before succeedingpolicy
```

To move a policy after another policy:

```bash
razor move-policy --name policy --after precedingpolicy
```

**enable-policy and disable-policy**

The `disable-policy` command prevents the specified policy from binding to any nodes. This can be useful if you want to temporarily inactivate a policy without deleting it. For example, you can use `disable-policy` to prevent nodes from installing a certain OS for a short period.

The `enable-policy` command reactivates a disabled policy.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to disable or enable.</td>
</tr>
</tbody>
</table>

**Sample commands**

To disable a policy:

```bash
razor disable-policy --name my_policy
```

To enable a policy:

```bash
razor enable-policy --name my_policy
```

**modify-policy-max-count**

The `modify-policy-max-count` command sets or removes the limit on the maximum number of nodes that can bind to a policy.

**Note:** To reduce the number of nodes a policy can bind to, you must use the `reinstall-node` command to mark nodes as uninstalled.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>max_count</td>
<td>*</td>
<td>2</td>
<td>Maximum number of nodes the policy can bind to.</td>
</tr>
<tr>
<td>no_max_count</td>
<td>*</td>
<td></td>
<td>Removes any limits on the number of nodes that can bind to the policy.</td>
</tr>
</tbody>
</table>

*You must specify one of the two count attributes, `max_count` or `no_max_count`.

**Sample commands**
To allow a policy to match an unlimited number of nodes:

```
razor --name example --no-max-count
```

To set a policy to match a maximum of 15 nodes:

```
razor --name example --max-count 15
```

**add-policy-tag**

The `add-policy-tag` command adds a new or existing tag to a policy. Because binding to policies happens only when unbound nodes check in, adding a policy tag doesn't affect nodes that are already bound to a policy.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>tag</td>
<td>#</td>
<td>2</td>
<td>Name of the tag to add to the policy.</td>
</tr>
<tr>
<td>rule</td>
<td></td>
<td>3</td>
<td>Creates a new tag using a case-sensitive match expression that applies the tag to a node if the expression evaluates as true. If tag creation fails, the policy isn't modified.</td>
</tr>
</tbody>
</table>

**Sample commands**

To add the existing tag `virtual` to the policy `example`:

```
razor add-policy-tag --name example --tag virtual
```

To add a new tag `virtual` to the policy `example`:

```
razor add-policy-tag --name example --tag virtual \ --rule '==,"fact", "virtual", "false"", "true"]'
```

**remove-policy-tag**

The `remove-policy-tag` command removes a tag from a policy. Because binding to policies happens only when unbound nodes check in, adding a policy tag doesn't affect nodes that are already bound to a policy.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>tag</td>
<td>#</td>
<td>2</td>
<td>Name of the tag to remove from the policy.</td>
</tr>
</tbody>
</table>

**Sample command**
To remove the tag virtual from the policy example:

```
razor remove-policy-tag --name example --tag virtual
```

**update-policy-repo**

The `update-policy-repo` command modifies the repository associated with a policy.

**Tip:** Before using this command, use `razor policies <POLICYNAME> nodes` to verify that no nodes are currently provisioning against the policy. If a node is provisioning against the policy when you run this command, provisioning might fail.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>repo</td>
<td>#</td>
<td>2</td>
<td>Name of the new repository associated with the policy.</td>
</tr>
</tbody>
</table>

**Sample command**

To update a policy's repository to a repository named fedora21:

```
razor update-policy-repo --policy my_policy --repo fedora21
```

**update-policy-task**

The `update-policy-task` command adds or removes a task from a policy.

**Tip:** Before using this command, use `razor policies <POLICYNAME> nodes` to verify that no nodes are currently provisioning against the policy. If a node is provisioning against the policy when you run this command, provisioning might fail.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>task</td>
<td>2</td>
<td></td>
<td>Name of the task to add to the policy.</td>
</tr>
<tr>
<td>no_task</td>
<td>true</td>
<td></td>
<td>true if the policy uses the task in the repository, or false.</td>
</tr>
</tbody>
</table>

**Sample commands**

To update a policy's task to a task named other_task:

```
razor update-policy-task --policy my_policy --task other_task
```

To use the task specified by the policy's repository:

```
razor update-policy-task --policy my_policy --no-task
```
**update-policy-broker**

The `update-policy-broker` command modifies the broker associated with a policy.

**Tip:** Before using this command, use `razor policies <POLICYNAME> nodes` to verify that no nodes are currently provisioning against the policy. If a node is provisioning against the policy when you run this command, provisioning might fail.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>broker</td>
<td>#</td>
<td>2</td>
<td>Name of the new broker associated with the policy.</td>
</tr>
</tbody>
</table>

**Sample command**

To update a policy's broker to a broker named `legacy-puppet`:

```
razor update-policy-broker --policy my_policy --broker legacy-puppet
```

**update-policy-node-metadata**

The `update-policy-node-metadata` command modifies the node metadata associated with a policy.

**Tip:** Before using this command, use `razor policies <POLICYNAME> nodes` to verify that no nodes are currently provisioning against the policy. If a node is provisioning against the policy when you run this command, provisioning might fail.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>#</td>
<td>1</td>
<td>Name of the policy to modify.</td>
</tr>
<tr>
<td>key</td>
<td>#</td>
<td>2</td>
<td>Name of the key to modify.</td>
</tr>
<tr>
<td>value</td>
<td>#</td>
<td>3</td>
<td>New value for the key.</td>
</tr>
<tr>
<td>no_replace</td>
<td></td>
<td></td>
<td><code>true</code> to cancel the update operation if the specified key is already present, or <code>false</code>.</td>
</tr>
</tbody>
</table>

**Sample command**

To update a policy's node metadata for the `my_key` value:

```
razor update-policy-node-metadata --policy policy1 --key my_key --value my_value
```

**delete-policy**

The `delete-policy` command deletes a policy from the Razor database. Nodes bound to the policy aren't re-provisioned.

**Command attributes**
Attributes | Required | Positional arguments | Description
--- | --- | --- | ---
name | # | 1 | Name of the policy to delete.

Sample command
To delete an obsolete policy:

```
razor delete-policy --name my_obsolete_policy
```

**Broker commands**
Broker commands enable you to create a new broker configuration, set or clear a specified key value for a broker, and delete a specified broker.

**create-broker**
The `create-broker` command creates a new broker configuration used to hand off management of nodes.

If you're using Puppet Enterprise for node management, use the `puppet-pe` broker type.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name for the new broker.</td>
</tr>
<tr>
<td>broker-type</td>
<td>#</td>
<td>2</td>
<td>Usually <code>puppet-pe</code>. Other valid values are <code>puppet</code>, to transfer control to an open source Puppet master, and <code>noop</code>, to leave the node unmanaged.</td>
</tr>
<tr>
<td>configuration</td>
<td></td>
<td></td>
<td>Configuration details specific to the <code>broker_type</code>. For <code>puppet_pe</code> brokers, properties include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• server — Host name of the Puppet master.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• version — The agent version to install. The default value is current.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ntpdate_server — URL for an NTP server, such as us.pool.ntp.org, used to synchronize the date and time before installing the Puppet agent.</td>
</tr>
</tbody>
</table>

Sample command
To create a simple puppet-pe broker:

```
razor create-broker --name puppet-pe -c server=puppet.example.org \
-c version=2015.3
```

**update-broker-configuration**

The `update-broker-configuration` command sets or clears a specified key value for a broker.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>broker</td>
<td>#</td>
<td>1</td>
<td>Name of the broker to update.</td>
</tr>
<tr>
<td>key</td>
<td>#</td>
<td>2</td>
<td>Name of the key to modify in the broker's configuration file. For puppet-pe brokers, this is usually server or version.</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td>3</td>
<td>New value to use for the key. This attribute can't be used with clear.</td>
</tr>
<tr>
<td>clear</td>
<td></td>
<td></td>
<td><code>true</code> to clear the value of the specified key, or <code>false</code>. This attribute can't be used with value.</td>
</tr>
</tbody>
</table>

**Sample command**

To change the key: `some_key` to `new_value`:

```
razor update-broker-configuration --broker mybroker --key some_key --value new_value
```

**delete-broker**

The `delete-broker` command deletes a specified broker from the Razor database.

**Note:** Before deleting a broker, remove any references to it in policies. This command fails if a policy is using the broker.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the broker to delete.</td>
</tr>
</tbody>
</table>

**Sample command**
To delete an obsolete broker:

```bash
razor delete-broker --name my_obsolete_broker
```

### Related information

- **Create a new broker type** on page 683

To create a broker called `sample`:

### Hook commands

Hooks are custom, executable scripts that are triggered to run when a node hits certain phases in its lifecycle. A hook script receives several properties as input, and can make changes in the node's metadata or the hook's internal configuration.

Hooks can be useful for:
- Notifying an external system about the stage of a node's installation.
- Querying external systems for information that modifies how a node gets installed.
- Calculating complex values for use in a node's installation configuration.

### `create-hook`

The `create-hook` command enables a hook to run when specified events occur.

#### Command attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>#</td>
<td>1</td>
<td>Name of the new hook.</td>
</tr>
<tr>
<td><code>hook_type</code></td>
<td>#</td>
<td>2</td>
<td>Type of hook that the new hook is based on. For available hook types on your server, run <code>razor create-hook --help</code>.</td>
</tr>
<tr>
<td><code>configuration</code></td>
<td></td>
<td></td>
<td>Configuration settings as required by the <code>hook_type</code>. This argument sets the initial configuration values for a hook. Configuration values can change as the hook is executed based on events or the <code>update-hook-configuration</code> command. This attribute can be abbreviated as <code>-c</code>.</td>
</tr>
</tbody>
</table>

#### Sample command

To create a simple hook:

```bash
razor create-hook --name myhook --hook-type some_hook --configuration foo=7
```
run-hook

The run-hook command executes a hook with parameters you specify. This command is useful to test a hook you're writing, or to re-run a hook that failed.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the hook to run.</td>
</tr>
<tr>
<td>event</td>
<td>#</td>
<td></td>
<td>Name of the event to trigger the hook.</td>
</tr>
<tr>
<td>node</td>
<td>#</td>
<td></td>
<td>Name of the node you want to run the hook on.</td>
</tr>
<tr>
<td>policy</td>
<td></td>
<td></td>
<td>Name of the policy you want to use as input to the hook script when the hook runs. This attribute applies only to events that affect policies, for example node-bound-to-policy and node-unbound-from-policy.</td>
</tr>
<tr>
<td>debug</td>
<td></td>
<td></td>
<td>true to include debug information in the event log, or false. If omitted, debug information is not logged.</td>
</tr>
</tbody>
</table>

**Sample command**

To run a hook called counter when a node boots:

```
razor run-hook --name counter --event node-booted --node node1
```

update-hook-configuration

The update-hook-configuration command sets or clears a specified key value for a hook. This can be useful to iteratively test a hook you're writing. For example, you can write the hook, test it with run-hook, then modify the hook as needed with update-hook-configuration.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hook</td>
<td>#</td>
<td>1</td>
<td>Name of the hook to update.</td>
</tr>
<tr>
<td>key</td>
<td>#</td>
<td>2</td>
<td>Name of the key to modify in the hook's configuration file.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Required</td>
<td>Positional arguments</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>value</td>
<td>*</td>
<td>3</td>
<td>Value to change the key to.</td>
</tr>
<tr>
<td>clear</td>
<td>*</td>
<td></td>
<td>true to clear the value of the specified key, or false.</td>
</tr>
</tbody>
</table>

* You must specify value or clear.

**Sample command**

To change the key some_key to new_value:

```
razor update-hook-configuration --hook hook1 \  
--key my_key --value new_value
```

**delete-hook**

The delete-hook command deletes a specified hook from the Razor database.

**Command attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Required</th>
<th>Positional arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#</td>
<td>1</td>
<td>Name of the hook to delete.</td>
</tr>
</tbody>
</table>

**Sample command**

To delete an obsolete hook:

```
razor delete-hook --name my_obsolete_hook
```

**Related information**

Available events

---

**Protecting existing nodes**

In *brownfield environments* – those in which you already have machines installed that PXE boot against the Razor server – you must take extra precautions to protect existing nodes. Failure to adequately protect existing nodes can result in data loss.

For recommended provisioning workflows in an existing environment, see Provisioning for advanced users.

**Protecting new nodes**

By default, Razor marks all newly discovered nodes as installed, which prevents modifications to the node. This default is controlled with the `protect_new_nodes` class parameter of the `pe_razor` class.

With `protect_new_nodes` enabled, Razor considers installed nodes eligible for reinstallation only when the installed flag is removed from the node using the `reinstall-node` command.

With `protect_new_nodes` disabled, Razor considers any nodes it detects – including installed nodes – eligible for provisioning. You might choose to disable `protect_new_nodes` if:

- You’re sure all nodes in your environment should be provisioned or reprovisioned.
- You’ve manually registered existing nodes that you want to protect.

The `protect_new_nodes` option is specified as a class parameter of the `pe_razor` class.
Related information
Register existing nodes manually on page 667
If you’re provisioning in an environment with existing nodes already installed, register the nodes to prevent Razor from re-provisioning them.

Reinstall the node on page 664
By default, Razor protects existing nodes from reprovisioning by marking all existing nodes as installed. You must specifically instruct the server to reinstall the node in order to trigger provisioning.

Change the protect_new_nodes default on page 668
Because you’ve already registered existing nodes to protect them from reprovisioning, it’s now safe to change the protect_new_nodes default to false. This removes the installed flag from unregistered nodes so that Razor can provision them.

Registering nodes
To identify existing nodes to the Razor server – and prevent reprovisioning – you can manually register nodes using the register-node command. The register-node command identifies a node as installed, which signals Razor to ignore the node.

To successfully register nodes, you must provide enough hw-info details for Razor to identify the nodes when they’re detected.

Related information
Register existing nodes manually on page 667
If you’re provisioning in an environment with existing nodes already installed, register the nodes to prevent Razor from re-provisioning them.

Limiting the number of nodes a policy can bind to
You can use the max_count attribute for policies to limit the number of slots available for provisioning.

For example, at initial installation, no slots are available, so no machines are provisioned. At this point, you can examine your resource pool or mark specific nodes as registered. If you create a new policy with a value of 1 for max_count, there’s now one slot available for provisioning. The first qualified node that checks in binds to the policy while all other nodes remain unprovisioned.

Provisioning a *nix node
Provisioning deploys and installs your chosen operating system to target nodes.

What triggers provisioning
There are four requirements for Razor to provision a node.

• The node must boot with iPXE software.
• The node’s network must link to the Razor server through TFTP.
• A Razor policy must match the node.
• The node’s installed flag must be set to false.

When these conditions are met, Razor recognizes the node, applies the first matching policy in the policy table, and provisions the node.

With these requirements in mind, you can modify your Razor workflow to suit your goals, your environment, and your familiarity with Razor.

• Provision for new users on page 661 enables you to learn about Razor and verify tags before provisioning nodes.
• Provision for advanced users on page 664 enables you to seamlessly provision nodes in an existing environment.
Provision for new users

This workflow enables you to learn about Razor and verify tags before provisioning nodes.

Before you begin

You're ready to provision a node after you configure:

• A DHCP/DNS/TFTP service with SELinux configured to enable PXE boot

• The server and client

To follow along with the examples in these workflows, you must have a new node with at least 1GB (2GB recommended) of memory. Don’t boot the node before you begin the provisioning process.

In this workflow, you load iPXE software and register nodes with the microkernel so you can view node details. Then you configure objects, finishing with creating a policy. Provisioning is triggered when you reinstall the node in order to remove the installed flag.

The examples in this workflow demonstrate provisioning a sample node with 6.7. You can modify the settings and scale up your workflow as needed for your environment.

Load iPXE software

Set your machines to PXE boot so that Razor can interact with the node and provision the operating system. This process uses both the undionly.kpxe file from the iPXE open source software stack and a Razorbootstrap.ipxe script.

1. Download the iPXE boot image undionly-20140116.kpxe and copy the image to your TFTP server’s /var/lib/tftpboot directory:

```
cp undionly-20140116.kpxe /var/lib/tftpboot
```

2. Download the iPXE bootstrap script from the server and copy the script to your TFTP server’s /var/lib/tftpboot directory:

```
```

Note: Don’t use localhost as the name of the host. The bootstrap script chain-loads the next iPXE script from the server, so it must contain the correct host name.

Register a node with the microkernel

Registering a node lets you learn about the node before Razor provisions it. With registered nodes, you can view facts about the node, add metadata to the node, and see which tags the node matches.

1. Boot the node. This can mean physically pressing the power button, using IPMI to manage the node’s power state, or, in the case of a VM, starting the VM.

   The node boots into the microkernel and Razor discovers the node. After the initial PXE boot, the node downloads the microkernel.

2. On the Razor server, view the new node: razor nodes

   The output displays the new node ID and name, for example:

   ```
   id: "http://localhost:8150/api/collections/node/node1"
   name: "node1"
   spec: "/razor/v1/collections/nodes/member"
   ```

   © 2020 Puppet, Inc.
3. View the node’s details: `razor nodes <NODE_NAME>`

The output displays hardware and DHCP information, the path to the log, and a placeholder for tags, for example:

```
   hw_info:
       mac: ["08-00-27-8a-5e-5d"]
     serial: "0"
       uuid: "9a717dc3-2392-4853-89b9-27feclaec7b2"
   dhcp_mac: "08-00-27-8a-5e-5d"
       log: log => http://localhost:8150/api/collections/node/node1/log
tags: []
```

4. When the microkernel is running on the machine, view facts about the node: `razor nodes <NODE_NAME> facts`

Facter periodically sends facts back to the server about the node, including IP address, details about network cards, and block devices.

**Create a repository**

Repositories contain – or point to – the operating system to install on a node.

You can create three types of repositories using specific attributes:

- `url` – Points to content available on another server, for example, on a mirror that you maintain.
- `iso-url` – Downloads and unpacks an ISO on the Razor server.
- `no_content` – Creates a stub directory on the Razor server that you can manually fill with content.

To download a CentOS 6.7 ISO and create a repository from it:

```
razor create-repo --name centos-6.7 --task centos
   --iso-url http://centos.sonn.com/6.7/isos/x86_64/CentOS-6.7-x86_64-bin-DVD1.iso
```

The ISO is downloaded onto the Razor server, then extracted to the repository. This can take some time to complete. To monitor progress, you can run `razor commands` to view the task status or `ls -al /tmp` to see the downloaded file size.

**Related information**

Repositories on page 676

A repository is where you store all of the actual bits used by Razor to install a node. Or, in some cases, the external location of bits that you link to. A repo is identified by a unique name.

Repository commands on page 642

Repository commands enable you to create and delete specified repositories from the Razor database and specify the task that installs the contents of a repository.

Repositories API on page 697

These commands enable you to create a new repository or delete a repository from the internal Razor database. You can also ensure that a specified repository uses a specified task.

**(Optional) Create tags**

Tags let you group nodes based on their characteristics. You can then apply policies based on tags to install appropriate operating systems on tagged nodes. If you don’t specify tags for a policy, the policy binds to any node.

1. To create a tag called `small` with a rule that matches machines that have less than 4GB of memory:

   ```
   razor create-tag --name small
   --rule '[:<, ["num", ["fact", "memorysize_mb"]], 4128]'
   ```
2. (Optional) Inspect the tag on the server: `razor tags <TAG_NAME>`
   For example, `razor tags small` responds with:

   ```
   From https://razor:8151/api/collections/tags/small:
   name: small
   rule: ["<", ["num", ["fact", "memorysize_mb"]], 4128]
   nodes: 1
   policies: 0
   ```

3. (Optional) Confirm that expected nodes now have the tag: `razor tags <TAG_NAME> nodes`
   For example, `razor tags small nodes` displays a table of registered nodes that have less than 4GB of memory.

   **Tip:** To see details about the policies associated with a tag, run `razor tags <TAG_NAME> policies`. To see its rule, run `razor tags <TAG_NAME> rule`.

### Related information

**Tags** on page 680
A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag's facts results in `true`. Tag matching is case sensitive.

**Tag commands** on page 646
Tag commands enable you to create new tags, set the rules used to apply the tag to nodes, change the rule of a specified tag, or delete a specified tag.

**Tags API** on page 699
These commands enable you to create a tag, delete a tag, or change the rule for a tag.

---

**Optional Create a broker**
Brokers hand off nodes to configuration management systems like Puppet Enterprise.

To hand off Razor nodes to a Puppet master at puppet-master.example.com:

```bash
razor create-broker --name pe --broker-type puppet-pe --configuration server=puppet-master.example.com
```

### Related information

**Brokers** on page 682
Brokers hand off nodes to configuration management systems like Puppet Enterprise. Brokers consist of two parts: a broker type and information specific to the broker type.

**Broker commands** on page 655
Broker commands enable you to create a new broker configuration, set or clear a specified key value for a broker, and delete a specified broker.

**Brokers API** on page 702
A broker is responsible for configuring a newly installed node for a specific configuration management system. For Puppet Enterprise, you generally only use brokers with the type `puppet-pe`.

---

**Add the pe_repo class to the PE Master node group**
To manage a node handed off by the broker, the master must include a class that matches the node’s architecture.

**Note:** Skip this step if the node you’re provisioning has the same architecture as your master, or if the master already includes a `pe_repo` class that matches the node’s architecture.

1. In the console, click **Classification**, and in the **PE Infrastructure** group, select the **PE Master** group.
2. On the **Configuration** tab in the **Add new class** field, select `pe_repo::platform::<VERSION>`.
   For example, `pe_repo::platform::el_7_x86_64`.
3. Click **Add class** and then commit changes.
4. On the master, run Puppet.

**Related information**

**Running Puppet on nodes** on page 364
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.

**Create a policy**
Policies tell Razor what OS to install on the provisioned node, where to get the OS software, how it should be configured, and how to communicate between the node and Puppet Enterprise.

To create a policy that installs CentOS on machines tagged with `small`, then hands them off to Puppet Enterprise for management:

```
razor create-policy --name centos-for-small
   --repo centos-6.7 --broker pe --tag small
   --hostname 'host${id}.example.com' --root-password secret
```

**Note:** You can view details of a specific policy by running `razor policies <POLICY_NAME>`. You can view a table of all policies by running `razor policies`. The order in which policies are listed in the table is important because Razor applies the first matching policy to a node.

**Related information**

**Policies** on page 681
Policies tell Razor what bits to install, where to get the bits, how they should be configured, and how the installed node can communicate with Puppet Enterprise.

**Policy commands** on page 648
Policies govern how nodes are provisioned.

**Policies API** on page 699
Policies govern how nodes are provisioned depending on how they are tagged.

**Reinstall the node**
By default, Razor protects existing nodes from reprovisioning by marking all existing nodes as installed. You must specifically instruct the server to reinstall the node in order to trigger provisioning.

You can skip this step if you change the `protect_new_nodes` option to `false`, which allows Razor to provision a node as soon as it PXE boots with a matching policy. Be sure you understand how the `protect_new_nodes` option works before changing it, however. Failure to protect existing nodes can result in data loss.

Reinstall the node: `razor reinstall-node <NODE_NAME>`

When you reinstall the node, Razor clears the `installed` flag and the node restarts and boots into the microkernel. The microkernel reports its facts, and Razor provisions the node by applying the first applicable policy in the policy table.

When provisioning is complete, you can log into the node using the `root_password` as specified by the node’s metadata, or by the policy that the node is bound to. You can also see the node and its details in the console, and manage it there as you would any other node.

**Provision for advanced users**
This workflow enables you to seamlessly provision nodes in an existing environment.

**Before you begin**
You're ready to provision a node after you configure:

- A DHCP/DNS/TFTP service with SELinux configured to enable PXE boot
The server and client

To follow along with the examples in these workflows, you must have a new node with at least 1GB (2GB recommended) of memory. Don’t boot the node before you begin the provisioning process.

In this workflow, you configure Razor objects, register any existing nodes to prevent accidentally overwriting them, and finally, load iPXE so that nodes boot through Razor. Provisioning is triggered when the node PXE boots with a matching policy in place.

The examples in this workflow demonstrate provisioning a sample node with 6.7. You can modify the settings and scale up your workflow as needed for your environment.

Create a repository

Repositories contain – or point to – the operating system to install on a node.

You can create three types of repositories using specific attributes:

- **url** – Points to content available on another server, for example, on a mirror that you maintain.
- **iso-url** – Downloads and unpacks an ISO on the Razor server.
- **no_content** – Creates a stub directory on the Razor server that you can manually fill with content.

To download a CentOS 6.7 ISO and create a repository from it:

```bash
razor create-repo --name centos-6.7 --task centos --iso-url http://centos.sonn.com/6.7/isos/x86_64/CentOS-6.7-x86_64-bin-DVD1.iso
```

The ISO is downloaded onto the Razor server, then extracted to the repository. This can take some time to complete. To monitor progress, you can run `razor commands` to view the task status or `ls -al /tmp` to see the downloaded file size.

Related information

Repositories on page 676

A repository is where you store all of the actual bits used by Razor to install a node. Or, in some cases, the external location of bits that you link to. A repo is identified by a unique name.

Repository commands on page 642

Repository commands enable you to create and delete specified repositories from the Razor database and specify the task that installs the contents of a repository.

Repositories API on page 697

These commands enable you to create a new repository or delete a repository from the internal Razor database. You can also ensure that a specified repository uses a specified task.

(Optional) Create tags

Tags let you group nodes based on their characteristics. You can then apply policies based on tags to install appropriate operating systems on tagged nodes. If you don’t specify tags for a policy, the policy binds to any node.

1. To create a tag called `small` with a rule that matches machines that have less than 4GB of memory:

   ```bash
   razor create-tag --name small --rule '["<", ["num", ["fact", "memorysize_mb"], 4128]]'
   ```

2. (Optional) Inspect the tag on the server: `razor tags <TAG_NAME>`

   For example, `razor tags small` responds with:

   ```json
   From https://razor:8151/api/collections/tags/small:
   name: small
   rule: ['<', ['num', ['fact', 'memorysize_mb']], 4128]
   nodes: 1
   policies: 0
   ```
3. **(Optional) Confirm that expected nodes now have the tag:** `razor tags <TAG_NAME> nodes`

   For example, `razor tags small nodes` displays a table of registered nodes that have less than 4GB of memory.

**Tip:** To see details about the policies associated with a tag, run `razor tags <TAG_NAME> policies`. To see its rule, run `razor tags <TAG_NAME> rule`.

**Related information**

**Tags** on page 680
A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag's facts results in true. Tag matching is case sensitive.

**Tag commands** on page 646
Tag commands enable you to create new tags, set the rules used to apply the tag to nodes, change the rule of a specified tag, or delete a specified tag.

**Tags API** on page 699
These commands enable you to create a tag, delete a tag, or change the rule for a tag.

**(Optional) Create a broker**
Brokers hand off nodes to configuration management systems like Puppet Enterprise.

To hand off Razor nodes to a Puppet master at `puppet-master.example.com`:

```
razor create-broker --name pe --broker-type puppet-pe
   --configuration server=puppet-master.example.com
```

**Related information**

**Brokers** on page 682
Brokers hand off nodes to configuration management systems like Puppet Enterprise. Brokers consist of two parts: a broker type and information specific to the broker type.

**Broker commands** on page 655
Broker commands enable you to create a new broker configuration, set or clear a specified key value for a broker, and delete a specified broker.

**Brokers API** on page 702
A broker is responsible for configuring a newly installed node for a specific configuration management system. For Puppet Enterprise, you generally only use brokers with the type `puppet-pe`.

**Add the pe_repo class to the PE Master node group**
To manage a node handed off by the broker, the master must include a class that matches the node’s architecture.

**Note:** Skip this step if the node you're provisioning has the same architecture as your master, or if the master already includes a `pe_repo` class that matches the node’s architecture.

1. In the console, click Classification, and in the PE Infrastructure group, select the PE Master group.
2. On the Configuration tab in the Add new class field, select `pe_repo::platform::<VERSION>`.

   For example, `pe_repo::platform::el_7_x86_64`.
3. Click Add class and then commit changes.
4. On the master, run Puppet.

**Related information**

**Running Puppet on nodes** on page 364

© 2020 Puppet, Inc.
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.

Create a policy
Policies tell Razor what OS to install on the provisioned node, where to get the OS software, how it should be configured, and how to communicate between the node and Puppet Enterprise.

To create a policy that installs CentOS on machines tagged with small, then hands them off to Puppet Enterprise for management:

```
razor create-policy --name centos-for-small
    --repo centos-6.7 --broker pe --tag small
    --hostname 'host${id}.example.com' --root-password secret
```

Note: You can view details of a specific policy by running `razor policies <POLICY_NAME>`. You can view a table of all policies by running `razor policies`. The order in which policies are listed in the table is important because Razor applies the first matching policy to a node.

Related information
Policies on page 681
Policies tell Razor what bits to install, where to get the bits, how they should be configured, and how the installed node can communicate with Puppet Enterprise.

Policy commands on page 648
Policies govern how nodes are provisioned.

Policies API on page 699
Policies govern how nodes are provisioned depending on how they are tagged.

Register existing nodes manually
If you’re provisioning in an environment with existing nodes already installed, register the nodes to prevent Razor from re-provisioning them.

You must provide enough hw-info details so that nodes can be identified when Razor detects them.

To register an existing node, and indicate with the `installed` attribute that the node isn’t eligible for provisioning:

```
razor register-node --hw-info net0=78:31:c1:be:c8:00 \
    --hw-info net1=72:00:01:f2:13:f0 \
    --hw-info net2=72:00:01:f2:13:f1 \
    --hw-info serial=xxxxxxxxxxx \
    --hw-info asset=Asset-1234567890 \
    --hw-info uuid="Not Settable" \
    --installed
```

Related information
Protecting existing nodes on page 659
In brownfield environments – those in which you already have machines installed that PXE boot against the Razor server – you must take extra precautions to protect existing nodes. Failure to adequately protect existing nodes can result in data loss.

Node commands on page 636
Nodes are created either through the node boot endpoint, when the node initiates its first web request to the Razor server, or through the register-node command.

Nodes API on page 693
These commands enable you to register a node, set a node's hardware information, remove a single node, or remove a node's associate with any policies and clear its installed flag.

**Change the protect_new_nodes default**
Because you've already registered existing nodes to protect them from reprovisioning, it's now safe to change the protect_new_nodes default to false. This removes the installed flag from unregistered nodes so that Razor can provision them.

1. In the console, select **Nodes > Classification**, then click the server node group.
2. On the **Configuration** tab, select the **protect_new_nodes** parameter, then in the **Value** field, enter **false**.
3. Commit changes, run , and then restart the **pe-razor-server** service.

**Related information**
**Running Puppet on nodes** on page 364
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.

**Load iPXE software**
Set your machines to PXE boot so that Razor can interact with the node and provision the operating system.
This process uses both the undionly.kpxe file from the iPXE open source software stack and a Razor-specific bootstrap.ipxe script.

1. Download the iPXE boot image **undionly-20140116.kpxe** and copy the image to your TFTP server’s `/var/lib/tftpboot` directory:
   ```bash
   `cp undionly-20140116.kpxe /var/lib/tftpboot`
   ```
2. Download the iPXE bootstrap script from the server and copy the script to your TFTP server’s `/var/lib/tftpboot` directory:
   ```bash
   ```
   **Note:** Don’t use localhost as the name of the host. The bootstrap script chain-loads the next iPXE script from the server, so it must contain the correct host name.
3. Boot the node. This can mean physically pressing the power button, using IPMI to manage the node’s power state, or, in the case of a VM, starting the VM.
   When the node PXE boots with a policy in place, Razor detects the node and provisions it by applying the first applicable policy in the policy table.
   When provisioning is complete, you can log into the node using the root_password as specified by the node’s metadata, or by the policy that the node is bound to. You can also see the node and its details in the console, and manage it there as you would any other node.

**Viewing information about nodes**
Use these commands to view details about nodes in your environment.

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>razor nodes</td>
<td>Displays a list of nodes that Razor knows about.</td>
</tr>
<tr>
<td>razor nodes &lt;NODE_NAME&gt;</td>
<td>Displays details about the specified node.</td>
</tr>
<tr>
<td>razor nodes &lt;NODE_NAME&gt; log</td>
<td>Displays a log that includes the timing and status of installation events, as well as downloads of kickstart files and post-install scripts.</td>
</tr>
</tbody>
</table>
Provisioning a Windows node

Provisioning deploys and installs your chosen operating system to target nodes.

What triggers provisioning

There are four requirements for Razor to provision a node.

- The node must boot with iPXE software.
- The node's network must link to the Razor server through TFTP.
- A Razor policy must match the node.
- The node's installed flag must be set to false.

When these conditions are met, Razor recognizes the node, applies the first matching policy in the policy table, and provisions the node.

With these requirements in mind, you can modify your Razor workflow to suit your goals, your environment, and your familiarity with Razor.

- Provision for new users on page 661 enables you to learn about Razor and verify tags before provisioning nodes.
- Provision for advanced users on page 664 enables you to seamlessly provision nodes in an existing environment.

Provision a Windows node

This Windows workflow adapts the provisioning for new users workflow. This process enables you to learn about Razor and verify tags before provisioning nodes.

Before you begin

You're ready to provision a node after you configure:

- A DHCP/DNS/TFTP service with SELinux configured to enable PXE boot
- The server and client

To provision Windows machines, you also need:

- A Windows machine running the same OS that you plan to provision. This machine is used to create a WinPE image.
- (Optional) An activation key for the OS. A trial license is used if you don't have an activation key.

To follow along with the examples in this workflow, you must also have a new node with at least 8GB of memory. Don’t boot the node before you begin the provisioning process.

In this workflow, you load iPXE software and register nodes with the microkernel so you can view node details. Then you configure objects, finishing with creating a policy. Provisioning is triggered when you reinstall the node in order to remove the installed flag.

The examples in this workflow demonstrate provisioning a sample node with Windows. You can modify the settings and scale up your workflow as needed for your environment.

Configure SMB share

Because neither the WinPE environment nor the Windows installer can use an HTTP source for installation, you must use a server message block (SMB) server to store the Razor repositories.

You can configure SMB share automatically, using a Razor class parameter, or manually.

Automatically configure SMB share

Enable the SMB share class parameter on the Razor server to let Razor set up the SMB share automatically.
1. In the console, select **Nodes > Classification**, then click the server node group.
2. On the **Configuration** tab, enable the SMB share parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_windows_smb</td>
<td>true</td>
</tr>
</tbody>
</table>

**Note:** If you change `enable_windows_smb` from **true** to **false** later, the share remains enabled but isn’t managed by Puppet.

3. Commit changes, run, and then restart the **pe-razor-server** service.

**Related information**

**Running Puppet on nodes** on page 364

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.

**Manually configure SMB share**

If Samba is already installed on your Razor server, you can manually configure it to work with Razor.

1. Navigate to the Samba directory: `cd /etc/samba`
2. Edit the `smb.conf` file:
   a) Modify the network settings as necessary for your environment.
   b) Edit the `global` service definition to allow unauthenticated access.

   ```
   [global]
   security = user
   map to guest = bad user
   ```

   c) Add a service definition that allows anonymous access and points to the `repo_store_root` class parameter of the `pe_razor` module.

   ```
   [razor]
   comment = Windows Installers
   path = /opt/puppetlabs/server/data/razor-server/repo
   guest ok = yes
   writable = no
   browsable = yes
   ```

3. Restart Samba: `service smb restart`

**Load iPXE software**

Set your machines to PXE boot so that Razor can interact with the node and provision the operating system. This process uses both the `undionly.kpxe` file from the iPXE open source software stack and a Razor bootstrap.ipxe script.

1. Download the iPXE boot image `undionly-20140116.kpxe` and copy the image to your TFTP server’s `/var/lib/tftpboot` directory:

   ```
   `cp undionly-20140116.kpxe /var/lib/tftpboot`
   ```
2. Download the iPXE bootstrap script from the server and copy the script to your TFTP server’s `/var/lib/tftpboot` directory:


   **Note:** Don’t use `localhost` as the name of the host. The bootstrap script chain-loads the next iPXE script from the server, so it must contain the correct host name.

**Register a node with the microkernel**

Registering a node lets you learn about the node before Razor provisions it. With registered nodes, you can view facts about the node, add metadata to the node, and see which tags the node matches.

1. Boot the node. This can mean physically pressing the power button, using IPMI to manage the node’s power state, or, in the case of a VM, starting the VM.

   The node boots into the microkernel and Razor discovers the node. After the initial PXE boot, the node downloads the microkernel.

2. On the Razor server, view the new node: `razor nodes`

   The output displays the new node ID and name, for example:

   ```
   id: "http://localhost:8150/api/collections/node/node1"
   name: "node1"
   spec: "/razor/v1/collections/nodes/member"
   ```

3. View the node’s details: `razor nodes <NODE_NAME>`

   The output displays hardware and DHCP information, the path to the log, and a placeholder for tags, for example:

   ```
   hw_info:
       mac: ["08-00-27-8a-5e-5d"]
       serial: "0"
       uuid: "9a717dc3-2392-4853-89b9-27feclae7b2"
   dhcp_mac: "08-00-27-8a-5e-5d"
   log:
       log => http://localhost:8150/api/collections/node/node1/log
   tags: []
   ```

4. When the microkernel is running on the machine, view facts about the node: `razor nodes <NODE_NAME> facts`

   Facter periodically sends facts back to the server about the node, including IP address, details about network cards, and block devices.

**Build a WinPE image**

Create a custom Windows Preinstallation Environment (WinPE) WIM image containing Razor scripts.

1. On an existing Windows machine running the same OS that you plan to install, install the Windows assessment and deployment kit in the default location.

2. Copy the directory at `/opt/puppetlabs/server/apps/razor-server/share/razor-server/build-winpe` from the Razor server to the existing Windows machine. If the directory is absent from your Razor server, use this archive.

3. Change into the directory you created in the previous step, for example, `c:\build-winpe`

4. (Optional) To include additional Windows drivers in the WIM image – for example, if your machines require proprietary UCS drivers – place `.inf` files in the `extra-drivers` folder inside the `build-winpe` directory.

   You can place individual files or directories in the `extra-drivers` folder.

© 2020 Puppet, Inc.
5. Run one of these build scripts, depending on whether you’re using unsigned drivers:
   
   - Standard build script
     
     ```powershell
     ```
   
   - Build script for unsigned drivers
     
     ```powershell
     ```

   When the build script completes, a `razor-winpe.wim` image appears in a new `razor-winpe` directory inside the current working directory.

### Create a repository and add the WinPE image

Because Razor can’t unpack Windows DVD images, you must create a stub repository and manually fill it with content.

1. Copy the Windows ISO image to the Razor server.
2. Create an empty repository.
   
   For example:
   
   ```bash
   razor create-repo --name win2012r2 --task windows/2012r2 --no-content true
   ```
3. After the repository is created, log into your Razor server as root, change into the repository directory, `/opt/puppetlabs/server/data/razor-server/<REPO_NAME>`, then mount the ISO image:
   
   ```bash
   $ mount -o loop </PATH/TO/WINDOWS_SERVER.ISO> /mnt
   $ cp -pr /mnt/* <REPO_NAME>
   $ umount /mnt
   $ chown -R pe-razor: <REPO_NAME>
   ```

   **Note:** The repository directory is specified by the `repo_store_root` class parameter of the `pe_razor` module. Your directory might be different from the default if you customized this parameter.
4. Copy the WinPE image from the existing Windows machine to the repository directory on the Razor server.

### Related information

- **Repositories** on page 676
  
  A repository is where you store all of the actual bits used by Razor to install a node. Or, in some cases, the external location of bits that you link to. A repo is identified by a unique name.

- **Repository commands** on page 642
  
  Repository commands enable you to create and delete specified repositories from the Razor database and specify the task that installs the contents of a repository.

- **Repositories API** on page 697
  
  These commands enable you to create a new repository or delete a repository from the internal Razor database. You can also ensure that a specified repository uses a specified task.

### (Optional) Create tags

Tags let you group nodes based on their characteristics. You can then apply policies based on tags to install appropriate operating systems on tagged nodes. If you don’t specify tags for a policy, the policy binds to any node.

1. To create a tag called `small` with a rule that matches machines that have less than 4GB of memory:
   
   ```bash
   razor create-tag --name small --rule \'"\"<", ["num", ["fact", "memorysize_mb"]], 4128\"\'"```
2. (Optional) Inspect the tag on the server: `razor tags <TAG_NAME>`

For example, `razor tags small` responds with:

```
From https://razor:8151/api/collections/tags/small:
  name: small
  rule: ["<", ["num", ["fact", "memorysize_mb"], 4128]
  nodes: 1
  policies: 0
```

3. (Optional) Confirm that expected nodes now have the tag: `razor tags <TAG_NAME> nodes`

For example, `razor tags small nodes` displays a table of registered nodes that have less than 4GB of memory.

**Tip:** To see details about the policies associated with a tag, run `razor tags <TAG_NAME> policies`. To see its rule, run `razor tags <TAG_NAME> rule`.

**Related information**

**Tags** on page 680
A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag's facts results in true. Tag matching is case sensitive.

**Tag commands** on page 646
Tag commands enable you to create new tags, set the rules used to apply the tag to nodes, change the rule of a specified tag, or delete a specified tag.

**Tags API** on page 699
These commands enable you to create a tag, delete a tag, or change the rule for a tag.

**(Optional) Create a broker**
Brokers hand off nodes to configuration management systems like Puppet Enterprise.

To hand off Razor nodes to a Puppet master at `puppet-master.example.com`:

```
razor create-broker --name pe --broker-type puppet-pe --configuration server=puppet-master.example.com
```

**Related information**

**Brokers** on page 682
Brokers hand off nodes to configuration management systems like Puppet Enterprise. Brokers consist of two parts: a broker type and information specific to the broker type.

**Broker commands** on page 655
Broker commands enable you to create a new broker configuration, set or clear a specified key value for a broker, and delete a specified broker.

**Brokers API** on page 702
A broker is responsible for configuring a newly installed node for a specific configuration management system. For Puppet Enterprise, you generally only use brokers with the type `puppet-pe`.

**Create a policy**
Policies tell Razor what OS to install on the provisioned node, where to get the OS software, how it should be configured, and how to communicate between the node and Puppet Enterprise.

To create a policy that installs CentOS on machines tagged with `small`, then hands them off to Puppet Enterprise for management:

```
razor create-policy --name centos-for-small --repo centos-6.7 --broker pe --tag small --hostname 'host${id}.example.com' --root-password secret
```
Note: You can view details of a specific policy by running `razor policies <POLICY_NAME>`. You can view a table of all policies by running `razor policies`. The order in which policies are listed in the table is important because Razor applies the first matching policy to a node.

**Related information**

*Policies* on page 681

Policies tell Razor what bits to install, where to get the bits, how they should be configured, and how the installed node can communicate with Puppet Enterprise.

*Policy commands* on page 648

Policies govern how nodes are provisioned.

*Policies API* on page 699

Policies govern how nodes are provisioned depending on how they are tagged.

**Reinstall the node**

By default, Razor protects existing nodes from reprovisioning by marking all existing nodes as installed. You must specifically instruct the server to reinstall the node in order to trigger provisioning.

You can skip this step if you change the `protect_new_nodes` option to `false`, which allows Razor to provision a node as soon as it PXE boots with a matching policy. Be sure you understand how the `protect_new_nodes` option works before changing it, however. Failure to protect existing nodes can result in data loss.

Reinstall the node: `razor reinstall-node <NODE_NAME>`

When you reinstall the node, Razor clears the `installed` flag and the node restarts and boots into the microkernel. The microkernel reports its facts, and Razor provisions the node by applying the first applicable policy in the policy table.

When provisioning is complete, you can log into the node using the `root_password` as specified by the node’s metadata, or by the policy that the node is bound to. You can also see the node and its details in the console, and manage it there as you would any other node.

**Viewing information about nodes**

Use these commands to view details about nodes in your environment.

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>razor nodes</code></td>
<td>Displays a list of nodes that Razor knows about.</td>
</tr>
<tr>
<td><code>razor nodes &lt;NODE_NAME&gt;</code></td>
<td>Displays details about the specified node.</td>
</tr>
<tr>
<td><code>razor nodes &lt;NODE_NAME&gt; log</code></td>
<td>Displays a log that includes the timing and status of installation events, as well as downloads of kickstart files and post-install scripts.</td>
</tr>
</tbody>
</table>

**Provisioning with custom facts**

You can use a microkernel extension to provision nodes based on hardware info or metadata that isn’t available by default in Facter.

When configured, all nodes receive the microkernel extension, which contains instructions for reporting custom facts to the Razor server. The custom facts can then be used to tag nodes, apply policies, and provision eligible nodes.

For example, if you want to provision machines based on hardware chassis or rack location – facts not available by default in Facter – you could create custom facts for these properties. Then, you can use the custom facts to create associated tags and policies that install the appropriate OS. For a detailed example of using a microkernel extension to report rack location, see Server locality using Razor and LLDP.
Creating custom facts for use with Razor is similar to creating custom facts for other Puppet uses. For more information about custom facts, see the Facter documentation.

**How the microkernel extension works**

On both new and existing nodes, the microkernel retrieves and unpacks the latest extension file before each checkin.

The content of the extension file is placed in a new, non-persistent directory on the microkernel image. Changes to the directory aren’t saved, and the directory is overwritten when a new extension file is available.

During unpacking, the executable bit on files is preserved. Permissions for the files in the extension are irrelevant, because the microkernel extension runs as root.

**Microkernel extension configuration**

The Razor microkernel extension is a ZIP file with the default title `mk-extension.zip`.

Depending on the requirements of the custom facts you’re using, the extension can include these directories:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contains</th>
<th>Environment variable modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>executables</td>
<td>PATH</td>
</tr>
<tr>
<td>lib</td>
<td>shared libraries</td>
<td>LD_LIBRARY_PATH</td>
</tr>
<tr>
<td>lib/ruby/facter/fact.rb</td>
<td>Ruby code</td>
<td>RUBYLIB</td>
</tr>
<tr>
<td>facts.d</td>
<td>external facts</td>
<td>–</td>
</tr>
</tbody>
</table>

For example, from inside your microkernal extension directory:

```
$ zip -r ../mk-extension.zip
bin/
    facts.d/
    facts.d/foobar.yaml
```

Note: You can’t change environment variables other than PATH, LD_LIBRARY_PATH, and RUBYLIB. For example, static FACTER_<factname> environment variables don’t work with the microkernel extension.

**Create the microkernel extension**

Create a microkernel extension ZIP file to customize the facts that are available for provisioning decisions.

1. Create a ZIP file, as described in Microkernel extension configuration, that contains the files required for your custom facts.
2. Place the ZIP file at `/etc/puppetlabs/razor-server` or, if you’ve changed the default, the location specified by the `microkernel_extension_zip` parameter of the `pe_razor` module.
3. Make sure the ZIP file is readable by the `pe-razor` user. For example, you can use `chmod 444 <FILE_NAME>` to make the file readable by all users.

**Tips and limitations of the microkernel extension**

Follow these guidelines to ensure that your microkernel extension behaves as expected.

- Use relative paths in your applications or facts, or search standard variables to locate content. Don’t use absolute paths, because the content of the directory that the microkernel extension is unpacked to is variable.
- To store persistent state, for example if you want to save the result of a web service lookup to avoid calling the service at each checkin, don’t use the location where the zip file is unpacked on the microkernel. Data stored at this location can be lost when the microkernel refreshes. Using `/tmp` is a valid choice, because the data persists in that location as long as the microkernel extension is running.

© 2020 Puppet, Inc.
Working with Razor objects

Provisioning with Razor requires certain objects that define how nodes are provisioned.

- **Repositories** on page 676
  A repository is where you store all of the actual bits used by Razor to install a node. Or, in some cases, the external location of bits that you link to. A repo is identified by a unique name.

- **Razor tasks** on page 677
  Razor tasks describe a process or collection of actions that are performed when Razor provisions machines. Tasks can be used to designate an operating system or other software to install, where to get it, and the configuration details for the installation.

- **Tags** on page 680
  A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag's facts results in true. Tag matching is case sensitive.

- **Policies** on page 681
  Policies tell Razor what bits to install, where to get the bits, how they should be configured, and how the installed node can communicate with Puppet Enterprise.

- **Brokers** on page 682
  Brokers hand off nodes to configuration management systems like Puppet Enterprise. Brokers consist of two parts: a broker type and information specific to the broker type.

- **Hooks** on page 684
  Hooks are an optional but useful Razor object. Hooks provide a way to run arbitrary scripts when certain events occur during a node's lifecycle. The behavior and structure of a hook are defined by a hook type.

- **Keeping Razor scalable** on page 689
  Speed provisioning and reduce load on the Razor server by following these scalability best practices.

Repositories

A repository is where you store all of the actual bits used by Razor to install a node. Or, in some cases, the external location of bits that you link to. A repo is identified by a unique name.

Instructions for the installation, such as what should be installed, where to get it, and how to configure it, are contained in tasks.

To load a repo onto the server, use the command:

```bash
razor create-repo --name=<repo name> --task <task name> --iso-url <URL>
```

For example:

```bash
razor create-repo --name centos-6.7 --task centos --iso-url http://centos.sonn.com/6.7/isos/x86_64/CentOS-6.7-x86_64-bin-DVD1.iso
```

There are three types of repositories that you might want to use, all created with the `create-repo` command:

- Repos where Razor downloads and unpacks ISOs for you and serves their contents.
- Repos that are external, such as a mirror that you maintain.
- Repos where a stub directory is created and you add the contents manually.

The `task` parameter is mandatory for creating all three of these types of repositories, and indicates the default installer to use with this repo. You can override a `task` parameter at the policy level. If you're not using a task, reference the stock task `noop`.

**Unpack an ISO and serve its contents**

This repository is created with the `--iso-url` property.
The server downloads and unpacks the ISO image onto its file system:

```bash
razor create-repo --name centos-6.7 --task centos --iso-url http://centos.sonn.com/6.7/isos/x86_64/CentOS-6.7-x86_64-bin-DVD1.iso
```

**Point to an existing resource**

To make a repository that points to an existing resource without loading anything onto the Razor server, provide a `url` property when you create the repository.

The `url` should be serving the unpacked contents of the install media.

```bash
razor create-repo --name centos-6.7 --task centos --url http://mirror.example.org/centos/6.7/
```

**Create a stub directory**

For some install media, especially Windows install DVDs, Razor is not able to automatically unpack the media; this is a known limitation of the library that Razor uses to unpack ISO images.

In those cases, it is necessary to first use `create-repo` to set up a stub directory on the Razor server, then manually add content to it. The stub directory is created with:

```bash
razor create-repo --name win2012r2 --task windows/2012r2 --no-content true
```

When this command completes successfully, log into your Razor server as root and `cd` into your server's repository directory. The repository directory is specified by the `repo_store_root` class parameter of the `pe_razor` class. By default, the directory is `/opt/puppetlabs/server/data/razor_server/repo`.

```
# mount -o loop /path/to/windows_server_2012_r2.iso /mnt
# cp -pr /mnt/* win2012r2
# umount /mnt
```

**Related information**

- **Create a repository** on page 662
- **Repositories contain – or point to – the operating system to install on a node.**
- **Repository commands** on page 642
- **Repository commands enable you to create and delete specified repositories from the Razor database and specify the task that installs the contents of a repository.**
- **Repositories API** on page 697
- **These commands enable you to create a new repository or delete a repository from the internal Razor database. You can also ensure that a specified repository uses a specified task.**

**Razor tasks**

Razor tasks describe a process or collection of actions that are performed when Razor provisions machines. Tasks can be used to designate an operating system or other software to install, where to get it, and the configuration details for the installation.

**Important:** tasks differ from tasks, which let you run arbitrary scripts and commands using . See the orchestrator documentation for information about tasks.

Tasks consist of a YAML metadata file and any number of ERB templates. Templates are used to generate things like the iPXE script that boots a node into the installer, and automated installation files like kickstart, preseed, or unattended files.

You specify the tasks you want to run in `policies`. 

© 2020 Puppet, Inc.
You can use several sources for tasks:

- Supported tasks that are included with Razor
- Copies of supported tasks that you customize as needed
- Tasks that you create from scratch

**Supported tasks**

Razor includes these supported tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentOS</td>
<td>6, 7</td>
<td>Enables deployment of clusters.</td>
</tr>
<tr>
<td>coreOS</td>
<td>1</td>
<td>System task. Boots the Razor microkernel.</td>
</tr>
<tr>
<td>Debian</td>
<td>wheezy</td>
<td>System task. Boots a system locally.</td>
</tr>
<tr>
<td>microkernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red Hat</td>
<td>6, 7</td>
<td></td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server</td>
<td>11, 12</td>
<td></td>
</tr>
<tr>
<td>Ubuntu</td>
<td>14.04, 16.04</td>
<td></td>
</tr>
<tr>
<td>VMware ESXi</td>
<td>5.5, 6</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>2008 R2, 2012 R2, 8 Pro, 2016</td>
<td></td>
</tr>
</tbody>
</table>

**Important:** Don’t modify supported tasks in place. If you want to customize a supported task, copy the task from the default task directory (`/opt/puppetlabs/server/apps/razor-server/share/razor-server/tasks`) to the custom task directory (`/etc/puppetlabs/razor-server/tasks`) before modifying the task.

**Storage directories**

Tasks are stored in the file system. The `task_path` class parameter of the `pe_razor` module determines where Razor looks for tasks.

The parameter can include a colon-separated list of paths. Relative paths in that list are taken to be relative to the top-level Razor directory. For example, setting `task_path` to `/opt/puppet/share/razor-server/tasks:/home/me/task:/tasks` makes Razor search for tasks in these three directories in order.

By default, there are two directories that store tasks:

- `/opt/puppetlabs/server/apps/razor-server/share/razor-server/tasks` stores default tasks shipped with the product.
- `/etc/puppetlabs/razor-server/tasks` stores custom tasks.

**Task metadata**

Tasks can include the following metadata in the task's YAML file. This file is called `metadata.yaml` and exists in `tasks/<NAME>.task` where `NAME` is the task name. Therefore, the task name looks like this: `tasks/<NAME>.task/metadata.yaml`.

```yaml
---
description: HUMAN READABLE DESCRIPTION
os: OS_NAME
os_version: OS_VERSION_NUMBER
base: TASK_NAME
boot_sequence:
```

© 2020 Puppet, Inc.
1: boot_templ1
2: boot_templ2
default: boot_local

Only os_version and boot_sequence are required. The base key allows you to derive one task from another by reusing some of the base metadata and templates. If the derived task has metadata that's different from the metadata in base, the derived metadata overrides the base task's metadata.

The boot_sequence hash indicates which templates to use when a node using this task boots. In the example above, a node first boots using boot_templ1, then using boot_templ2. For every subsequent boot, the node uses boot_local.

**Task templates**

Task templates are ERB templates and are searched in all the directories in the task_path configuration setting.

Templates are searched in the subdirectories in this order:

1. name.task
2. base.task if present
3. common

**Template helpers**

Templates can use the following helpers to generate URLs that point back to the server; all of the URLs respond to a GET request, even the ones that make changes on the server.

- **task**: Includes attributes such as name, os, os_version, boot_seq, label, description, base, and architecture.
- **node**: Includes attributes such as name, metadata, and facts.

**Tip**: You can use node.hw_hash['fact_boot_type'] == "efi" to evaluate whether a node booted via UEFI.

- **repo**: Includes attributes such as name, iso_url, url.
- **file_url(TEMPLATE, RAW)**: The URL that retrieves TEMPLATE.erb (after evaluation) from the current node's task. By default, the file is interpolated (RAW=false). If the file doesn't need to be interpolated, specify RAW=true.
- **repo_url(PATH)**: The URL to the file at PATH in the current repo.
- **repo_file_contents(PATH)**: The contents of the file at PATH inside the repo. This is an empty string if the file does not exist.
- **repo_file?(PATH)**: Whether a file exists at the given path. This is nil if the file does not exist.
- **log_url(MESSAGE, SEVERITY)**: The URL that logs MESSAGE in the current node's log.
- **node_url**: The URL for the current node.
- **store_url(VARS)**: The URL that stores the values in the hash VARS in the node. Currently only changing the node's IP address is supported. Use store_url("ip" => "192.168.0.1") for that.
- **stage_done_url**: The URL that tells the server that this stage of the boot sequence is finished, and that the next boot sequence should begin upon reboot.
- **broker_install_url**: A URL from which the install script for the node's broker can be retrieved. You can see an example in the script, os_complete.erb, which is used by most tasks.

Each boot (except for the default boot) must culminate in something akin to curl <%= stage_done_url %>
before the node reboots. Omitting this causes the node to reboot with the same boot template over and over again.

The task must indicate to the Razor server that it has successfully completed by doing a GET request against stage_done_url("finished"), for example using curl or wget. This marks the node installed in the Razor database.

You use these helpers by causing your script to perform an HTTP GET against the generated URL. This might mean that you pass an argument like ks=<%= file_url("kickstart") %> when booting a kernel, or that you put curl <%= log_url("Things work great") %> in a shell script.

© 2020 Puppet, Inc.
Related information

Task commands on page 645
Use task commands to create tasks in the Razor database.

Tasks API on page 698
This commands enables you to create a task in the Razor database.

Tags

A tag consists of a unique name and a rule. Tags match a node if evaluating the node against the tag’s facts results in true. Tag matching is case sensitive.

For example, to create a tag, small, that matches any machine with less than 4GB of memory:

```
razor create-tag --name small
--rule '["<", ["num", ["fact", "memorysize_mb"], 4128]]
```

Tag rules

Rule expressions are of the form op arg1 arg2 ... argn where op is one of the accepted operators, and arg1 through argn are the arguments for the operator. If the arguments are expressions themselves, they’re evaluated before op.

Here are some example tag rules:

• To match nodes with more than 10 processors: [">", ["num", ["fact", "processorcount"], 10]
• To match nodes with specified MAC addresses: ["has_macaddress", "de:ea:db:ee:f0:00", "de:ea:db:ee:f0:01"]

Tag operators

The expression language supports these operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Returns</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>[&quot;=&quot;, arg1, arg2]</td>
<td>The arg1 fact for the node. The optional arg2 is used if the arg1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fact isn't present.</td>
<td></td>
</tr>
<tr>
<td>[&quot;!=&quot; arg1, arg2]</td>
<td>True if the specified arguments are equal.</td>
<td>&quot;eq&quot;</td>
</tr>
<tr>
<td>[&quot;and&quot;, arg1, ..., argn]</td>
<td>True if all arguments are true.</td>
<td></td>
</tr>
<tr>
<td>[&quot;or&quot;, arg1, ..., argn]</td>
<td>True if any argument is true.</td>
<td></td>
</tr>
<tr>
<td>[&quot;not&quot;, arg]</td>
<td>True if the argument evaluates to false or nil.</td>
<td></td>
</tr>
<tr>
<td>[&quot;fact&quot;, arg1 (, arg2)]</td>
<td>The optional arg2 is used if the arg1 fact isn't present.</td>
<td></td>
</tr>
<tr>
<td>[&quot;metadata&quot;, arg1 (,</td>
<td>The arg metadata entry for the node. The optional arg2 is used if the</td>
<td></td>
</tr>
<tr>
<td>arg2)]</td>
<td>arg1 fact isn't present.</td>
<td></td>
</tr>
<tr>
<td>[&quot;tag&quot;, arg]</td>
<td>True if the node has the specified tag.</td>
<td></td>
</tr>
<tr>
<td>[&quot;has_macaddress&quot;, arg1,</td>
<td>True if any facts that start with &quot;macaddress&quot; matches one of arg1</td>
<td></td>
</tr>
<tr>
<td>arg2 ..., argn]</td>
<td>... argn.</td>
<td></td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
### Related information

**(Optional) Create tags** on page 662

Tags let you group nodes based on their characteristics. You can then apply policies based on tags to install appropriate operating systems on tagged nodes. If you don’t specify tags for a policy, the policy binds to any node.

**Tag commands** on page 646

Tag commands enable you to create new tags, set the rules used to apply the tag to nodes, change the rule of a specified tag, or delete a specified tag.

**Tags API** on page 699

These commands enable you to create a tag, delete a tag, or change the rule for a tag.

### Policies

Policies tell Razor what bits to install, where to get the bits, how they should be configured, and how the installed node can communicate with Puppet Enterprise.

Policies can contain tags, which match characteristics of nodes to specific policies. For example, you might create a policy that installs a certain operating system on nodes greater than 5GB in memory.

Policies without tags bind to any node. You might create policies without tags if:

- You have a single policy that installs the same operating system on all nodes.
- You want to install a specific number of various operating systems on a number of undifferentiated nodes. In this case, you can use the `max-count` attribute to specify how many of each operating system to provision.

For example, to create a policy, **centos-for-small**, that is applied to the first 20 nodes that match the `small` tag:

```bash
razor create-policy --name centos-for-small --repo centos-6.7 --broker pe --tag small --hostname 'host${id}.example.com' --root-password secret --max-count 20
```
How policies bind to nodes

When a node boots into the Razor microkernel, it sends its facts to the Razor server. The node then binds to the first policy in the policy table that applies to the node. When a node binds to a policy, the node is provisioned with the operating system specified by the policy.

If the node doesn't match any policies, it continues to send facts periodically to the Razor server and remains un provisioned until it does match a policy.

Policies don't bind to nodes if:

- The policy is disabled.
- The policy has already reached the maximum number of nodes that can bind to it.
- The policy requires tags that don't apply to the node.

Important: If you don't manage policies carefully, you can inadvertently enable Razor to match with and provision machines that you don't want to provision. In the case of existing servers, this can lead to catastrophic data loss.

The policy table

Policies are stored in a policy table. The order of the policy table is important because Razor applies the first policy that matches to a node.

You can influence the order of policies by:

- Using the create-policy command with before or after parameters to indicate where the new policy should appear in the policy table.
- Using the move-policy command with before and after parameters to reorder existing policies.

Related information

Create a policy on page 664

Policies tell Razor what OS to install on the provisioned node, where to get the OS software, how it should be configured, and how to communicate between the node and Puppet Enterprise.

Policy commands on page 648

Policies govern how nodes are provisioned.

Policies API on page 699

Policies govern how nodes are provisioned depending on how they are tagged.

Node metadata commands on page 641

Node metadata commands enable you to update, modify, or remove metadata from nodes.

Brokers

Brokers hand off nodes to configuration management systems like Puppet Enterprise. Brokers consist of two parts: a broker type and information specific to the broker type.

Razor ships with three default broker types:

- puppet-pe — Hands off node management to Puppet Enterprise. This broker specifies the address of the Puppet server, the Puppet Enterprise version, and for Windows, the location of the Windows agent installer.
- puppet — Hands off management to open source Puppet. This broker specifies the address of the Puppet server, the node certname, and the environment.
- noop — Doesn't hand off management. A no-op broker can be useful for getting started quickly or doing a basic installation without configuration management.

You can create brokers using the create-broker command. You can also update configuration details for the broker and delete brokers.

Writing new broker types is only necessary to use Razor with other configuration management systems.
Broker storage directories

There are two directories that store brokers:

- /opt/puppetlabs/server/apps/razor-server/share/razor-server/brokers stores default brokers shipped with the product.
- /etc/puppetlabs/razor-server/brokers stores custom brokers that you create.

Tip: We recommend not modifying the directory or brokers at /opt..., but you can copy brokers from there to the custom broker directory and modify them as needed.

Creating a PE broker

To create a PE broker that enrolls nodes with the master at puppet-master.example.com:

```bash
razor create-broker --name=my_puppet --broker-type=puppet-pe
  --configuration server=puppet.example.org
  --configuration version=2015.3
```

Writing the broker install script

The broker install script is generated from the install.erb (*nix) or install.ps1.erb (Windows) template of your broker.

The template returns a valid shell script because tasks generally perform the handoff to the broker by running a command like `curl -s <%= broker_install_url %> | /bin/bash`. The GET request to `broker_install_url`(*nix) or `broker_install_url('install.ps1')` (Windows) returns the broker’s install script after interpolating the template.

In the install template, you have access to two objects: node and broker.

The node object gives you access to node facts (`node.facts['example']`), tags (`node.tags`), and metadata (`node.metadata['key']`).

The broker object gives you access to the configuration settings. For example, if your `configuration.yaml` specifies that a setting `version` must be provided when creating a broker from this broker type, you can access the value of `version` for the current broker as `broker.version`.

Writing the broker configuration file

The `configuration.yaml` file indicates what parameters can be supplied for any given broker type.

For each parameter, you can supply these attributes:

- description — Human-readable description of the parameter.
- required — true to indicate that the parameter must be supplied. Parameters that aren’t required are optional.
- default — Value for the parameter if one isn’t supplied.

As an example, here's the `configuration.yaml` for the PE broker type:

```
server:
  description: "The puppet master to load configurations and installation packages from."
version:
  description: "Override the PE version to install; defaults to ‘current’.
windows_agent_download_url:
  description: "The download URL for a Windows PE agent installer; defaults to a URL derived from the ‘version’ config."
```

Create a new broker type

To create a broker called sample:
1. From the command line, create a sample.broker directory anywhere on the broker_path.

   The broker_path is specified in the broker_path class parameter of the pe_razor class. By default, the broker directory for custom brokers is /etc/puppetlabs/razor-server/brokers.

2. Write the broker install script and place it in the install.erb (*nix) or install.ps1.erb (Windows PowerShell) template in the sample.broker directory.

3. If the broker type requires configuration data, write the broker configuration file and save it in the sample.broker directory.

**Related information**

(Optional) Create a broker on page 663

Brokers hand off nodes to configuration management systems like Puppet Enterprise.

**Broker commands** on page 655

Broker commands enable you to create a new broker configuration, set or clear a specified key value for a broker, and delete a specified broker.

**Brokers API** on page 702

A broker is responsible for configuring a newly installed node for a specific configuration management system. For Puppet Enterprise, you generally only use brokers with the type puppet-pe.

**Hooks**

Hooks are an optional but useful Razor object. Hooks provide a way to run arbitrary scripts when certain events occur during a node's lifecycle. The behavior and structure of a hook are defined by a hook type.

The two primary components for hooks are:

- **Configuration** — A JSON document for storing data on a hook. Configurations have an initial value and can be updated by hook scripts.
- **Event scripts** — Scripts that run when a specified event occurs. Event scripts must be named according to the handled event.

**Hook storage directories**

There are two directories that store hooks:

- /opt/puppetlabs/server/apps/razor-server/share/razor-server/hooks stores default hooks shipped with the product.
- /etc/puppetlabs/razor-server/hooks stores custom hooks.

**Note:** Don't modify the directory or hooks at /opt..., but you can copy hooks from there to the custom hook directory and modify them as needed.

**File layout for a hook type**

Similar to brokers and tasks, hook types are defined through a .hook directory and optional event scripts within that directory.

```
hooks/
    some.hook/
        configuration.yaml
        node-bind-policy
        node-unbind-policy
    ...
```

**Available events**

These are the events that you can create hooks for.

- **node-booted** — Triggered every time a node boots via iPXE.
• node-registered — Triggered after a node has been registered. Limited hardware information is available after registration.
• node-deleted — Triggered after a node has been deleted.
• node-bound-to-policy — Triggered after a node has been bound to a policy. The script input contains a policy property with the details of the policy that has been bound to the node.
• node-unbound-from-policy — Triggered after a node has been marked as uninstalled by the reinstall-node command and thus has been returned to the set of nodes available for installation.
• node-facts-changed — Triggered whenever a node changes its facts.
• node-install-finished — Triggered when a policy finishes its last step.

Creating hooks
The create-hook command is used to create a hook object from a hook type. For example:

```bash
gerazor create-hook --name myhook --hook-type some_hook --configuration example1=7 --configuration example2=rhubarb
```

The hook object created by this command sets its initial configuration to the JSON document:

```json
{  "example1": 7,  "example2": "rhubarb"
}
```

Each time an event script for a hook runs, it has an opportunity to modify the hook's configuration. These changes to the configuration are preserved by the Razor server. The server also makes sure that hooks don't modify their configurations concurrently in order to avoid data corruption.

The delete-hook command is used to remove a hook.

Hook configuration
Hook scripts can use the hook object's configuration.

The hook type specifies the configuration data that it accepts in configuration.yaml. That file must define a hash:

```yaml
example1:  
  description: "Explain what example1 is for"  
  default: 0
example2:  
  description "Explain what example2 is for"  
  default: "Barbara"
...
```

For each event that the hook type handles, it must contain a script with the event's name. That script must be executable by the Razor server. All hook scripts for a certain event are run in an indeterminate order when that event occurs.

Event scripts
The general protocol is that hook event scripts receive a JSON object on their stdin, and might return a result by printing a JSON object to their stdout.

The properties of the input object vary by event, but they always contain a hook property:

```json
{  "hook": {    "name": hook name,    "configuration": ... operations to perform ...
```

© 2020 Puppet, Inc.
The `configuration` object is initialized from the hash described in the hook's `configuration.yaml` and the properties set by the current values of the hook object's `configuration`. With the `create-hook` command above, the input JSON would be:

```json
{
    "hook": {
        "name": "myhook",
        "configuration": {
            "update": {
                "example1": 7,
                "example2": "rhubarb"
            }
        }
    }
}
```

The script might return data by producing a JSON object on its `stdout` to indicate changes that should be made to the hook's `configuration`. The updated `configuration` is used on subsequent invocations of any event for that hook. The output must indicate which properties to update, and which ones to remove:

```json
{
    "hook": {
        "configuration": {
            "update": {
                "example1": 8
            },
            "remove": [ "frob" ]
        }
    }
}
```

The Razor server ensures that invocations of hook scripts are serialized. For any hook, events are processed one-by-one to allow for transactional safety around the changes any event script might make.

### Node events

Most events are directly related to a node. The JSON input to the event script has a `node` property that contains the representation of the node in the same format the API produces for node details.

The JSON output of the event script can modify the node metadata:

```json
{
    "node": {
        "metadata": {
            "update": {
                "example1": 8
            },
            "remove": [ "frob" ]
        }
    }
}
```

### Error handling

The hook script must exit with exit code 0 if it succeeds; any other exit code is considered a failure of the script.

Whether the failure of a script has any other effects depends on the event. A failed execution can still make updates to the hook and node objects by printing to `stdout` in the same way as a successful execution.

To report error details, the script should produce a JSON object with an `error` property on its `stdout` in addition to exiting with a non-zero exit code. If the script exits with exit code 0, the `error` property is still recorded, but the
event's severity isn't an error. The error property should itself contain an object whose message property is a human-readable message; additional properties can be set. For example:

```json
{
    "error": {
        "message": "connection refused by frobnicate.example.com",
        "port": 2345,
        ...
    }
}
```

**Sample input**

The input to the hook script is in JSON, containing a structure like this:

```json
{
    "hook": {
        "name": "counter",
        "configuration": { "value": 0 }
    },
    "node": {
        "name": "node10",
        "hw_info": {
            "mac": [ "52-54-00-30-8e-45" ],
            ...
        },
        "dhcp_mac": "52-54-00-30-8e-45",
        "tags": [ "compute", "anything", "any", "new" ],
        "facts": {
            "memorysize_mb": "995.05",
            "facterversion": "2.0.1",
            "architecture": "x86_64",
            ...
        },
        "state": {
            "installed": false
        }},
    "hostname": "client-l.watzmann.net",
    "root_password": "secret",
    "netmask_eth0": "255.255.255.0",
    "ipaddress_lo": "127.0.0.1",
    "last_checkin": "2014-05-21T03:45:47+02:00"
},
"policy": {
    "name": "client-l",
    "repo": "centos-6.7",
    "task": "ubuntu",
    "broker": "noop",
    "enabled": true,
    "hostname_pattern": "client-l.watzmann.net",
    "root_password": "secret",
    "tags": [ "client-l" ],
    "nodes": { "count": 0 }
}
```

**Sample hook**

Here is an example of a basic hook called **counter** that counts the number of times Razor registers a node.
This example creates a corresponding directory for the hook type, `counter.hook`, inside the `hooks` directory. You can store the current count as a configuration entry with the key `count`. Thus the `configuration.yaml` file might look like this:

```yaml
count:
  description: "The current value of the counter"
  default: 0
```

To make sure a script runs whenever a node is bound to a policy, create a file called `node-bound-to-policy` and place it in the `counter.hook` folder. Then write this script, which reads in the current configuration value, increments it, then returns some JSON to update the configuration on the hook object:

```
#!/bin/bash

json=$(< /dev/stdin)
name=$(jq ".hook.name" <<< $json)
value=$(( $(jq ".hook.config.count" <<< $json) + 1 ))

cat <<EOF
  "hook": {
    "configuration": {
      "update": {
        "count": $value
      }
    }
  },
  "node": {
    "metadata": {
      "$name": $value
    }
  }
}
EOF
```

Note that this script uses `jq`, a bash JSON manipulation framework. This must be on the `$PATH` in order for execution to succeed.

Next, create the hook object, which stores the configuration:

```
razor create-hook --name counter --hook-type counter
```

Since the configuration is absent from this creation call, the default value of 0 in `configuration.yaml` is used. Alternatively, this could be set using `--configuration count=0` or `--c count=0`.

The hook is now ready to use. You can query the existing hooks in a system via `razor hooks`. To query the current value of the hook's configuration, `razor hooks counter` will show `count` initially set to 0. When a node gets bound to a policy, the `node-bound-to-policy` script is triggered, yielding a new configuration value of 1.

**Assign dynamic hostnames using hooks**
You can use a hook to create more advanced dynamic hostnames than the simple incremented pattern — `\{id \}.example.com` — from the `hostname` property on a policy.

**Before you begin**
Ruby must be installed in `$PATH` for the hook script to succeed. By default, Ruby is installed as required with Puppet Enterprise. If Ruby isn't installed, add it to one of the paths specified in the `hook_execution_path` class parameter of the `pe_razor` class.

© 2020 Puppet, Inc.
This type of hook calculates the correct hostname and returns that hostname as metadata on the node. To do so, it uses a basic counter system that stores the number of nodes bound to a given policy.

This hook is intended to be extended for cases where an external system needs to be contacted to determine the correct hostname. In such a scenario, the new value is still returned as metadata for the node.

1. Create an instance of a default hook:

   ```bash
   razor create-hook --name some_policy_hook --hook-type hostname 
   --configuration policy=some_policy 
   --configuration hostname-pattern='$(policy)${count}.example.com'
   ```

2. (Optional) If multiple policies require their own counter, create multiple instances of this hook with different policy or hostname-pattern hook configurations.

Running the create-hook command kicks off this sequence of events:

1. The counter for the policy starts at 1.
2. When a node boots, the node-bound-to-policy event is triggered.
3. The policy's name from the event is then passed to the hook as input.
4. The hook matches the node's policy name to the hook's policy name.
5. If the policy matches, the hook calculates a rendered hostname-pattern:
   - It replaces ${count} with the current value of the counter hook configuration.
   - It left-pads the ${count} with padding zeroes. For example, if the hook configuration's padding equals 3, a count of 7 will be rendered as 007.
   - It replaces ${policy} with the name of its policy.
6. The hook returns the rendered hostname-pattern as the node metadata of hostname and returns the incremented value for the counter that was used, so that the next execution of the hook uses the next value.

Viewing the hook's activity log

To view the status of the hook's executions, see razor hooks $name log:

   ```markdown
   timestamp: 2015-04-01T00:00:00-07:00
   policy: policy_name
   cause: node-bound-to-policy
   exit_status: 0
   severity: info
   actions: updating hook configuration: {"update":{"counter=>2}} and
            updating node metadata:
            {"update":{"hostname=""policy_name1.example.com"}}
   ```

Related information

Hook commands on page 657
Hooks are custom, executable scripts that are triggered to run when a node hits certain phases in its lifecycle. A hook script receives several properties as input, and can make changes in the node's metadata or the hook's internal configuration.

Hooks API on page 703
Hooks are custom, executable scripts that are triggered to run when a node hits certain phases in its lifecycle. A hook script receives several properties as input, and can make changes in the node’s metadata or the hook’s internal configuration.

Keeping Razor scalable

Speed provisioning and reduce load on the Razor server by following these scalability best practices.

- When creating repos, use url rather than iso-url. If you have a mirror hosting installation files, they can be distributed straight to the node rather than having the Razor server distribute the files.
• When matching policies, limit the number of tags you use.
  
  • Create tags using the has_macaddress operator to identify a list of nodes to be assigned the tag, for example, ["has_macaddress", "de:ea:db:ee:f0:00", "de:ea:db:ee:f0:01"].
  
  • Set the policy metadata key on nodes using the modify-node-metadata command, then create matching tags, for example, ["=". ["metadata", "policy"], "policy1"]').
  
• When running hooks, avoid long- and frequent-running scripts, which all run on the same thread synchronously.

Using the Razor API

The Razor API is REST-based.

By default, API calls are sent over HTTPS with TLS/SSL. The default URL for the API entrypoint is https://localhost:8151/api. This URL varies if you're running the Razor client on a different machine than the server, or if you changed the default port due to a port conflict.

The API uses JSON exclusively, and all requests must set the HTTP Content-Type header to application/json and must Accept application/json in the response.

The Razor API is stable, and clients can expect operations that work against this version of the API to work against future versions of the API. However, we might add information or functionality to the API, so clients must ignore anything in responses they receive from the server that they don't understand.

Structure and keys

Everything underneath /api is part of the public API and is stable.

Note: The /svc namespace is an internal namespace used for communication with the iPXE client, the microkernel, and other internal components of Razor. This namespace is not enumerated under /api and has no stability guarantee. We recommend making /svc URLs available only to that part of the network that contains nodes that need to be provisioned.

The top-level /api entrypoint serves as the start for navigating through the Razor command and query facilities. For example, given the default API URL, call GET https://razor:8151/api to view the API.

The response is a JSON object with the following keys:

• commands — The commands available on this server.
• collections — Read-only queries available on this server.
• version — The version of Razor that is running on the server. The version should only be used for diagnostic purposes and for bug reporting, and never to decide whether the server supports a certain operation or not.

Each of those keys contains a JSON array, with a sequence of JSON objects that have the following keys:

• name — A human-readable label for an object, usually unique only among objects of the same type on the same server.
• id — The URL of an entity. A GET request against a URL with an id attribute produces a representation of the object.
• rel — A ”spec URL” that indicates the type of data contained. Use this key to discover the endpoint that you want to follow, rather than using the name.

For example, {"name": "add-policy-tag", "rel": "https://api.puppetlabs.com/razor/v1/commands/add-policy-tag", "id": "https://localhost:8151/api/commands/add-policy-tag"}. 
© 2020 Puppet, Inc.
Commands

The list of commands that the Razor server supports is returned as part of a request to GET /api in the commands array. Clients can identify commands using the rel attribute of each entry in the array, and should make their POST requests to the URL given in the id attribute.

The id URL for each command supports the following HTTP methods:

- GET — Retrieve information about the command, such as a help text and machine-readable information about the parameters this command takes.
- POST — Execute the command. Command parameters are supplied in a JSON document in the body of the POST request.

Commands are generally asynchronous and return a status code of 202 Accepted on success. The response from a command generally has this form:

```json
{
   "result":"Policy win2012r2 disabled",
   "command":"http://razor:8088/api/collections/commands/74"
}
```

Here, result is a human-readable explanation of what the command did, and command points into the collection of all the commands that were ever run against this server. Performing a GET against the command URL provides additional information about the execution of this command, such as the status of the command, the parameters sent to the server, and details about errors.

Tip: Most client commands allow positional arguments, which can save keystrokes.

Related information

Using positional arguments with Razor client commands on page 634

Most Razor client commands allow positional arguments, which means that you don’t have to explicitly enter the name of the argument, like --name. Instead, you can provide the values for each argument in a specific order.

Collections

In addition to the supported commands above, a GET /api request returns a list of supported collections in the collections array.

Each entry contains at minimum the following keys:

- name — A human-readable name for the collection.
- id — The endpoint through which the collection can be retrieved (via GET).
- rel — The type of the collection.

A GET request to the id of a collection returns a JSON object. The spec property of that object indicates the type of collection. The total indicates how many items there are in the collection in total (not just how many were returned by the query). The items value is the actual list of items in the collection, a JSON array of objects. Each object has these properties:

- id — A URL that uniquely identifies the object. A GET request to this URL provides further detail about the object.
- spec — A URL that identifies the type of the object.
- name — A human-readable name for the object.

Object details

Performing a GET request against the id of an item in a collection returns further detail about that object. Different types of objects provide different properties.

For example, here is a sample tag listing:

```json
{
```
References to other resources are represented as a single JSON object (in the case of a one-to-one relationship) or an array of JSON objects (for a one-to-many or many-to-many relationship). Each JSON object contains these fields:

- **id** — A URL that uniquely identifies the associated object or collection of objects.
- **spec** — The type of the associated object.
- **name** — A human-readable name for the object.
- **count** — The number of objects in the associated collection.

### Querying the node collection

You can query nodes based on `hostname` or fields stored in `hw_info`.

- **hostname** — A regular expression to match against hostnames. The results include partial matches, so `hostname=example` returns all nodes whose hostnames include example.
- **fields stored in hw_info** — `mac`, `serial`, `asset`, and `uuid`.

For example, the following queries the UUID to return the associated node:

```
/api/collections/nodes?uuid=9ad1e079-b9e3-347c-8b13-9b42cbf53a14
```

```
{
  "items": [
    {
      "id": "https://razor.example.com:8151/api/collections/nodes/node14",
      "name": "node14",
      "spec": "http://api.puppetlabs.com/razor/v1/collections/nodes/member"
    }
  ],
  "spec": "http://api.puppetlabs.com/razor/v1/collections/nodes"
}
```

### Paging collections

The `nodes` and `events` collections are paginated.

GET requests for them may include the following parameters to limit the number of items returned:

- **limit** — Only return this many items.
- **start** — Return items starting at `start`. 
Razor API reference

Use the Razor API to interact with Razor and provision bare metal nodes

The default bootstrap iPXE file
A GET request to /api/microkernel/bootstrap returns the iPXE script that you should put on your TFTP server (usually called bootstrap.ipxe). The script gathers information about the node (the hw_info) that it sends to the server and that the server uses to identify the node and determine how exactly the node should boot.

The URL accepts the parameter nic_max, which you should set to the maximum number of network interfaces that respond to DHCP on any given node. It defaults to 4.

The URL also accepts an http_port parameter, which tells Razor which port its internal HTTP communications should use, and the /svc URLs must be available through that port. The default install uses 8150 for this.

Configuration API
Razor configuration is pulled from a configuration file controlled by Puppet Enterprise. You can change configuration values in the console with class parameters of the pe_razor class.

Note: In order for configuration changes to take effect, you must restart the Razor service by running service pe-razor-server restart on the Razor server.

View configuration (config)
The config endpoint displays details about your Razor configuration.

Note: Properties specified in the api_config_blacklist aren't returned by the config endpoint.

The endpoint handles a GET request to the collection URL specified in /api.

Nodes API
These commands enable you to register a node, set a node's hardware information, remove a single node, or remove a node's associate with any policies and clear its installed flag.

Register a node (register-node)
Register a node with Razor before it is discovered, and potentially provisioned.

In environments in which some nodes have been provisioned outside of the purview of Razor, this command offers a way to tell Razor that a node is valuable and not eligible for automatic reprovisioning. Such nodes are only reprovisioned if they are marked available with the reinstall-node command.

The register-node command allows you to perform the same registration that would happen when a new node checks in, but ahead of time. The register-node command uses the installed value to indicate that a node has already been installed, which signals to Razor that the node should be ignored, and Razor should act as if it had successfully installed that node.

In order for this command to be effective, hw_info must contain enough information that the node can successfully be identified based on the hardware information sent from iPXE when the node boots; this usually includes the MAC addresses of all network interfaces of the node.

register-nodes accepts the following parameters:

• hw_info — Required, provides the hardware information for the node. This is used to match the node on first boot with the record in the database. The hw_info can contain all or a subset of the following entries:
  • netN — The MAC addresses of each network interface, for example net0 or net2: The order of the MAC addresses is not significant.
  • serial — The DMI serial number of the node.
  • asset — The DMI asset tag of the node.
  • uuid — DMI UUID of the node.
  • installed — A boolean flag indicating whether this node should be considered installed and therefore not eligible for reprovisioning by Razor
API example

Registering a machine before booting it with installed set to true protects it from accidental reinstallation by Razor:

```json
{
    "hw_info": {
        "net0": "78:31:c1:be:c8:00",
        "net1": "72:00:01:f2:13:f0",
        "net2": "72:00:01:f2:13:f1"
    },
    "installed": true
}
```

Set node hardware info (set-node-hw-info)

When a node's hardware changes, such as a network card being replaced, the Razor server needs to be informed so it can correctly match the new hardware to the existing node definition.

The set-node-hw-info command lets you replace the existing hardware data with new data, prior to booting the modified node on the network. For example, update node172 with new hardware information as follows:

```json
{
    "node": "node172",
    "hw_info": {
        "net0": "78:31:c1:be:c8:00",
        "net1": "72:00:01:f2:13:f0",
        "net2": "72:00:01:f2:13:f1"
    }
}
```

The format of the hw_info is the same as for the register-node command.

Delete node (delete-node)

To remove a single node, provide its name:

```json
{
    "name": "node17"
}
```

Note: If the deleted node boots again at some point, Razor automatically recreates it.

Reinstall node (reinstall-node)

To remove a node's association with any policy and clear its installed flag, provide its name:

```json
{
    "name": "node17"
}
```

Once the node reboots, it boots back into the microkernel, goes through discovery and tag matching, and can bind to another policy for reinstallation. This command does not change the node's metadata or facts.

IPMI API

IPMI commands are node commands based on the Intelligent Platform Management Interface.

Note: You must install the ipmitool on the Razor server before using IPMI commands. To install the tool, run yum install ipmitool -y.
Set node IPMI credentials (**set-node-ipmi-credentials**)

Razor can store IPMI credentials on a per-node basis. These credentials include a hostname (or IP address), username, and password to use when contacting the BMC/LOM/IPMI LAN or LANplus service to check or update power state and other node data.

After IPMI credentials have been set up for a node, you can use the **reboot-node** and **set-node-desired-power-state** commands.

These three data items can only be set or reset together, in a single operation. When you omit a parameter, Razor sets it to NULL (representing no value, or the NULL username/password as defined by IPMI).

The structure of a request is:

```json
{
    "name": "node17",
    "ipmi_hostname": "bmc17.example.com",
    "ipmi_username": null,
    "ipmi_password": "sekretskwirrl"
}
```

This command only works with remote IPMI targets, not locally; therefore, you **must** provide an IPMI hostname.

**Reboot node (**reboot-node**)**

If you've associated IPMI credentials with a node, Razor can use IPMI to trigger a hard power cycle.

Just provide the name of the node:

```json
{
    "name": "node1"
}
```

The IPMI communication spec includes some generous internal rate limits to prevent it from overwhelming the network or host server. If an execution slot isn't available on the target node, your **reboot-node** command goes into a background queue, and runs as soon as a slot is available.

This background queue is cumulative and persistent: there are no limits on how many commands you can queue up, how frequently a node can be rebooted, or how long a command can stay in the queue. If you restart your Razor server before the queued commands are executed, they'll remain in the queue and run after the server restarts.

The **reboot node** command is not integrated with IPMI power state monitoring, so you can't see power transitions in the record or when polling the node object.

**Set a node's desired power state (**set-node-desired-power-state**)**

By default, Razor checks your nodes' power states every few minutes in the background. If it detects a node in a non-desired state, Razor issues an IPMI command directing the node to its desired state.

To set the desired state for a node:

```json
{
    "name": "node1234",
    "to":   "on"|"off"|null
}
```

The **name** parameter identifies the node to change the setting on. The **to** parameter contains the desired power state to set. Valid values are **on**, **off**, or **null** (the JSON NULL/nil value), which reflect "power on", "power off", and "do not enforce power state" respectively.

© 2020 Puppet, Inc.
Node metadata API
These commands enable you to add, update, or remove metadata keys and remove metadata entries.

Modify node metadata (modify-node-metadata)
Node metadata is a collection of key/value pairs, much like a node’s facts. The difference is that the facts represent what the node tells Razor about itself, while its metadata represents what you tell Razor about the node.

The modify-node-metadata command lets you add, update, or remove individual metadata keys, or clear a node’s metadata:

```
{
  "node": "node1",
  "update": {
    "key1": "value1",
    "key2": "value2",
    ...
  },
  "remove": [ "key3", "key4", ... ],
  "no_replace": true
}
```

or

```
{
  "node": "node1",
  "clear": true
}
```

You can submit multiple updates or removals in a single command. However, clear only works on its own.

Tip: In batch updates with no_replace, use force to bypass errors. Existing keys aren’t modified.

Update node metadata (update-node-metadata)
The update-node-metadata command offers a simplified way to update a single metadata key.

The body for the command must be:

```
{
  "node": "node1",
  "key": "my_key",
  "value": "my_val",
  "no_replace": true
}
```

The no_replace parameter is optional. If it is true, the metadata entry will not be modified if it already exists.

Remove node metadata (remove-node-metadata)
The remove-node-metadata command offers a simplified way to remove either a single metadata entry or all metadata entries on a node:

```
{
  "node": "node1",
  "key": "my_key",
}
```
Repositories API
These commands enable you to create a new repository or delete a repository from the internal Razor database. You can also ensure that a specified repository uses a specified task.

Create new repository (create-repo)
The `create-repo` command creates a new repository. The repository can contain the content to install a node, or it can point to an existing online repository.

You can create three types of repositories:

- Those that reference content available on another server, for example, on a mirror you maintain (url).
- Those where unpacks ISOs for you and serves their contents (iso_url).
- Those where creates a stub directory that you can manually fill with content (no_content).

The `task` parameter is mandatory in all three variants of this command, and indicates the default task that should be used when installing machines using this repository. The `task` parameter can be overridden at the policy level. If you're not using a task, reference the stock task `noop`.

To have Razor unpack an ISO for you and serve its content:

```json
{
  "name": "fedora19",
  "iso_url": "file:///tmp/Fedora-19-x86_64-DVD.iso",
  "task": "puppet"
}
```

**Tip:** Supplying the `iso_url` property when you create a repository ensures that you can delete it from the server with the `delete-repo` command.

To create a repository that points to an existing resource without loading anything onto the Razor server:

```json
{
  "name": "fedora19",
  "url": "http://mirrors.n-ix.net/fedora/linux/releases/19/Fedora/x86_64/os/"
  "task": "noop"
}
```

To create an empty directory that you can manually fill later:

```json
{
  "name": "win2012r2",
  "no_content": true
  "task": "noop"
}
```

After creating a `no_content` repository, you can log into your Razor server and fill the repository directory with the content, for example by loopback-mounting the install media and copying it into the directory. The repository directory is specified by the `repo_store_root` class parameter of the `pe_razor` class. By default, the directory is in `/opt/puppetlabs/server/data/razor_server/repo`.

Using `no_content` is usually required for Windows install media, because the library that Razor uses to unpack ISO images can't handle Windows ISO images.
Delete a repository (delete-repo)

The delete-repo command deletes a repository from the internal Razor database.

The command accepts a single repository name:

```json
{
   "name": "fedora16"
}
```

This command deletes the repository from the internal Razor database. If you supplied the iso_url property when you created the repository, the folder is also deleted from the server. If you didn't supply the iso_url property, content remains in the repository directory.

Update a repository's specified task (update-repo-task)

Ensures that a specified repository uses the task this command specifies, setting the task if necessary. If a node is currently provisioning against the repo when you run this command, provisioning might fail.

```json
{
   "repo": "my_repo",
   "task": "other_task"
}
```

Tasks API

This commands enables you to create a task in the Razor database.

Important: tasks differ from tasks, which let you run arbitrary scripts and commands using . See the orchestrator documentation for information about tasks.

Create task (create-task)

The create-task command creates a task in the Razor database. This command is an alternative to manually placing task files in the task_path. If you anticipate needing to make changes to tasks, we recommend the disk-backed task approach.

The body of the POST request for this command has the following form:

```json
{
   "name": "redhat6",
   "os": "Red Hat Enterprise Linux",
   "boot_seq": {
      "1": "boot_install",
      "default": "boot_local"
   },
   "templates": {
      "boot_install": " ... ERB template for an ipxe boot file ...",
      "installer": " ... another ERB template ..."
   }
}
```

The possible properties in the request are:

- name — The name of the task; must be unique.
- os — The name of the OS; mandatory.
- description — Human-readable description.
- boot_seq — A hash mapping the boot counter or 'default' to an ERB template.
- templates — A hash mapping template names to the actual ERB template text.
Tags API
These commands enable you to create a tag, delete a tag, or change the rule for a tag.

Create tag (create-tag)
To create a tag, use the following in the body of your POST request:

```json
{
    "name": "small",
    "rule": ["=",["fact", "processorcount"], "2"]
}
```

The name of the tag must be unique; the rule is a match expression.

Delete tag (delete-tag)
To delete a tag, use the following in the body of your POST request:

```json
{
    "name": "small",
    "force": true
}
```

You can't delete a tag while it's being used by a policy, unless you set the optional force parameter to true. In that case, Razor removes the tag from all policies using it and then deletes it.

Update tag (update-tag-rule)
To change the rule for a tag, use the following in the body of your POST request:

```json
{
    "name": "small",
    "rule": ["<=",["fact", "processorcount"], "2"],
    "force": true
}
```

This changes the rule of the given tag to the new rule. Razor then reevaluates the tag against all nodes and updates each node's tag attribute to reflect whether the tag now matches or not.

If the tag is used by any policies, the update is only performed if you set the optional force parameter to true. Otherwise, the command returns status code 400.

Policies API
Policies govern how nodes are provisioned depending on how they are tagged.

Razor maintains an ordered table of policies. When a node boots, Razor traverses this table to find the first eligible policy for that node. A policy might be ineligible for binding to a node if the node does not contain all of the tags on the policy, if the policy is disabled, or if the policy has reached its maximum for the number of allowed nodes.

When you list the policies collection, the list is in the order in which Razor checks policies against nodes.

Create policy (create-policy)

```json
{
    "name": "a policy",
    "repo": "some_repo",
    "task": "redhat6",
    "broker": "puppet",
    "hostname": "host${id}.example.com",
    "root_password": "secret",
    "max_count": 20,
```
"before" | "after": "other policy",
"node_metadata": { "key1": "value1", "key2": "value2" },
"tags": ["existing_tag", "another_tag"]
}

**Note:** Because the policy contains many fields, you might want to put it in a JSON file. If you do, then your `create-policy` command would include the file name, like this: `razor create-policy --json <name of policy file>.json`.

Tags, repos, tasks, and brokers are referenced by name. The `tags` are optional. A policy with no tags can be applied to a node. The `task` is also optional. If it is omitted, the repo's task is used. The `repo` and `broker` entries are required.

The `hostname` parameter defines a simple pattern for the hostnames of nodes bound to your policy. The `${id}` references each node's DB id.

The `max_count` parameter sets an upper limit on how many nodes can be bound to your policy at a time. You can specify a positive integer, or make it unlimited by setting it to `nil`.

Razor considers each policy sequentially, based on its order in a table. By default, new policies go at the end of the table. To override the default order, include a `before` or `after` argument referencing an existing policy by name.

The `node_metadata` parameter lets your policy apply metadata to a node when it binds. This does not overwrite existing metadata; it only adds keys that are missing. To install Windows on non-English systems, specify the `win_language` using the culture code for the appropriate Microsoft language pack). For example, `--node-metadata win_language=es-ES`.

**Move policy (move-policy)**

This command lets you change the order in which Razor considers your policies for matching against nodes.

To move an existing policy into a different place in the order, use the `move-policy` command with a body like:

```json
{
    "name": "a policy",
    "before" | "after": "other policy"
}
```

This changes the policy table so that "a policy" appears before or after "other policy".

**Enable/disable policy (enable-policy/disable-policy)**

To keep a policy from being matched against any nodes, disable it with the `disable-policy` command.

To enable a disabled policy, use the `enable-policy` command. Both commands use a body like:

```json
{
    "name": "a policy"
}
```

**Modify the max_count for a policy (modify-policy-max-count)**

The command `modify-policy-max-count` lets you set the maximum number of nodes that can be bound to a specific policy.

The body of the request should be of the form:

```json
{
    "name": "a policy",
    "max_count": new-count
}
```
new-count can be an integer, which must be greater than the number of nodes that are currently bound to the policy. Alternatively, the **no_max_count** argument makes the policy unbounded:

```
{
   "name": "a policy",
   "no_max_count": true
}
```

**Add tags to policy (add-policy-tag)**

To add tags to a policy, supply the name of a policy and of the tag:

```
{
   "name": "a-policy-name",
   "tag" : "a-tag-name",
}
```

To create the tag in addition to adding it to the policy, supply the **rule** argument:

```
{
   "name": "a-policy-name",
   "tag" : "a-new-tag-name",
   "rule": "new-match-expression"
}
```

**Remove tags from policy (remove-policy-tag)**

To remove tags from a policy, supply the name of a policy and the name of the tag:

```
{
   "name": "a-policy-name",
   "tag" : "a-tag-name",
}
```

A policy with no tags can still be applied to any node.

**Update a policy's specified repository (update-policy-repo)**

Ensures that a policy uses the repository this command specifies. If necessary, **update-policy-repo** sets the repository, for example if a policy has already been created and you want to add a repository to it.

The following shows how to update a policy's repository to a repository called "fedora21":

```
{
   "node": "node1",
   "policy": "my_policy",
   "repo": "fedora21"
}
```

**Update a policy's specified task (update-policy-task)**

Ensures that a policy uses the task this command specifies. If necessary, **update-policy-task** sets the task, for example if a policy has already been created and you want to add a task to it.

If a node is currently provisioning against the policy when you run this command, provisioning can fail.

The following shows how to update a policy's task to a task called "other_task".

```
{
   "node": "node1",
```
Update a policy's specified broker (**update-policy-broker**)

Ensures that a policy uses the broker this command specifies. If necessary, **update-policy-broker** sets the broker, for example if a policy has already been created and you want to add a broker to it.

If a node is currently provisioning against the policy when you run this command, provisioning can fail.

The following shows how to update a policy's broker to a broker called "legacy-puppet":

```json
{
    "node": "node1",
    "policy": "my_policy",
    "broker": "legacy-puppet"
}
```

Update a policy's node metadata (**update-policy-node-metadata**)

Ensures that a policy uses the node metadata this command specifies. If necessary, **update-policy-node-metadata** sets the node metadata, for example if a policy has already been created and you want to add node metadata to it.

The following shows how to update a policy's node metadata for the "my_key" value:

```json
{
    "node": "node1",
    "policy": "my_policy",
    "key": "my_key",
    "value": "my_value"
}
```

Delete policy (**delete-policy**)

To delete a policy, supply the name of a single policy:

```json
{
    "name": "my-policy"
}
```

Note that this does not affect the **installed** status of a node, and therefore can't, by itself, make a node bind to another policy upon reboot.

**Brokers API**

A broker is responsible for configuring a newly installed node for a specific configuration management system. For Puppet Enterprise, you generally only use brokers with the type **puppet-pe**.

Create broker (**create-broker**)

To create a broker, post the following to the **create-broker** URL:

```json
{
    "name": "puppet",
    "configuration": {
        "server": "puppet.example.org"
    },
    "broker-type": "puppet-pe"
}
```
The broker-type must correspond to a broker that is present in a valid broker directory. Broker directories are specified in the `broker_path` class parameter of the `pe_razor` class. By default, the broker path for custom brokers is `/etc/puppetlabs/razor-server/brokers:brokers`.

The permissible settings for the configuration hash depend on the broker type and are declared in the broker type's `configuration.yaml`. For the `puppet-pe` broker type, these are:

- **server** — The hostname of the master.
- **version** — The agent version to install; this defaults to `current` and should only be used in exceptional circumstances.
- **ntupdate_server** — URL for an NTP server, such as `us.pool.ntp.org`, used to synchronize the date and time before installing the agent.

**Update broker configuration (update-broker-configuration)**

To set or clear a specified key value for a broker, use `update-broker-configuration`.

This argument changes the key "some_key" to "new_value":

```json
{
    "broker": "mybroker",
    "key": "some_key",
    "value": "new_value"
}
```

This argument clears the value for "some_key":

```json
{
    "broker": "mybroker",
    "key": "some_key",
    "clear": true
}
```

The `update-broker-configuration` command can be useful to update the Puppet Enterprise version of a `puppet-pe` broker. For example, if you created a `puppet-pe` broker for version `2015.2.0`, you can update it with:

```json
{
    "broker": "puppet-pe",
    "key": "version",
    "value": "2015.3.0"
}
```

**Delete broker (delete-broker)**

The `delete-broker` command only requires the name of the broker:

```json
{
    "name": "small"
}
```

It is not possible to delete a broker that is used by a policy.

**Hooks API**

Hooks are custom, executable scripts that are triggered to run when a node hits certain phases in its lifecycle. A hook script receives several properties as input, and can make changes in the node’s metadata or the hook’s internal configuration.

Hooks can be useful for:

- Notifying an external system about the stage of a node’s installation.
- Querying external systems for information that modifies how a node gets installed.
- Calculating complex values for use in a node’s installation configuration.

Create hook (create-hook)

To create a new hook, use the create-hook command with a body like this:

```json
{
    "name": "myhook",
    "hook_type": "some_hook",
    "configuration": {
        "foo": 7,
        "bar": "rhubarb"
    }
}
```

The hook_type must correspond to a hook with the specified name in a valid hook directory. Hook directories are specified in the hook_path class parameter of the pe_razor class. By default, the hook directory for custom hooks is /etc/puppetlabs/razor-server/hooks:hooks.

The optional configuration parameter lets you provide a starting configuration corresponding to that hook_type.

Update hook configuration (update-hook-configuration)

To set or clear a specified key value for a hook, use update-hook-configuration.

This argument changes the key "some_key" to "new_value":

```json
{
    "hook": "myhook",
    "key": "some_key",
    "value": "new_value"
}
```

This argument clears the value for "some_key":

```json
{
    "hook": "myhook",
    "key": "some_key",
    "clear": true
}
```

Delete hook (delete-hook)

To delete a single hook, provide its name:

```json
{
    "name": "my-hook"
}
```

Upgrading Razor

If you used Razor in a previous Puppet Enterprise environment, upgrade Razor to keep your Puppet Enterprise and Razor versions synched.

After a Puppet Enterprise upgrade, the pe_razor class continues to operate normally, but we recommend upgrading Razor as soon as possible to avoid unintended effects.

Note: If nodes are actively provisioning during upgrade, provisioning might fail. You can resume provisioning after pe-razor-server restarts.
Upgrade Razor from Puppet Enterprise 2015.2.x or later

Upgrading from 2015.2.x or later is a mostly automated process that replaces the software repository, installs software packages, and migrates the Razor database. Upgrading to 2016.2 or later requires manual migration of any custom configuration from your config.yaml file to parameters in the `pe_razor` class.

Before you begin

If you’re upgrading to Puppet Enterprise 2016.2 or later and you’ve modified your `config.yaml` file – for example, by changing `protect_new_nodes` or customizing tasks, brokers, or hooks – make a note of the modified settings.

For instructions on upgrading Razor from Puppet Enterprise 3.8, see earlier versions of the Razor documentation.

1. Upgrade the master.
2. Upgrade the agent on the Razor server node.
   
   The `pe-razor-server` service automatically restarts.

3. (Optional) If you’re upgrading to 2016.2 or later, transfer any customized configurations to parameters in the `pe_razor` class.

   **Note:** To prevent accidentally overwriting machines during upgrade, the default for `protect_new_nodes` was changed to `true` in Puppet Enterprise 2016.2 and later. If your environment and workflows rely on provisioning all new nodes, you must manually change `protect_new_nodes` to `false` after upgrade, then run puppet and restart the `pe-razor-server` service.

   **Tip:** You can run `razor --version` to verify that the upgrade was successful.

Related information

Running Puppet on nodes on page 364

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.

Uninstalling Razor

If you’re permanently done provisioning nodes, you can uninstall Razor.

Uninstall the Razor server

To uninstall the Razor server, erase Razor from the server node and update your environment so that nodes continue to boot as expected.

1. On the node with the Razor server, run `yum erase pe-razor`

2. Drop the PostgreSQL database that the server used.

3. Change DHCP/TFTP so that the machines that have been installed continue to boot outside the scope of Razor.

Uninstall the Razor client

Uninstall the Razor client if you no longer wish to use it to interact with a Razor server.

Run `gem uninstall razor-client`. 

© 2020 Puppet, Inc.
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.

- **Regenerating certificates: monolithic installs** on page 706
  In some cases, you may find that you need to regenerate the certificates and security credentials (private and public keys) generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master that you need to move to a different network in your infrastructure, or you may find that you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

- **Regenerating certificates: split installs** on page 710
  In some cases, you may find that you need to regenerate the SSL certificates and security credentials (private and public keys) that are generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master you need to move to a different network in your infrastructure, or you may find you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

- **Individual PE component cert regeneration (split installs only)** on page 712
  If you encounter a security vulnerability, or need to change your certificates for some other reason (for example, if you have a hostname change), you can regenerate the certificate and security credentials (private and public keys) generated for the PE components.

- **Regenerate Puppet agent certificates** on page 717
  From time to time, you might encounter a situation in which you need to regenerate a certificate for an agent. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you're receiving SSL errors on your agent that are preventing you from performing normal operations.

- **Regenerate compile master certs** on page 718
  From time to time, you may encounter a situation in which you need to regenerate a certificate for a compile master. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you're receiving strange SSL errors on your compile master that are preventing you from performing normal operations.

- **Use the PE CA as an intermediate CA** on page 719
  PE can operate as an intermediate CA to an external root CA. (It can't be an intermediate to an intermediate.) In this configuration, the PE CA is left enabled and it can automatically accept CSRs, distribute certificates to agents, and use the standard `puppet cert` command to sign certificates.

- **Use a custom SSL certificate for the console** on page 721
  The 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. You might find that this is not an acceptable scenario and want to use a custom CA to create the console's certificate.

- **Change the hostname of a monolithic master** on page 722
  To change the hostnames assigned to your PE infrastructure nodes requires updating the corresponding PE certificate names. You might have to update your master hostname, for example, when migrating to a new PE installation, or after an organizational change such as a change in company name.

- **Generate a custom Diffie-Hellman parameter file** on page 723
  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.

- **Disable TLSv1 in PE** on page 724
  You can disable TLSv1 in PE to comply with standards as necessary.

---

**Regenerating certificates: monolithic installs**

In some cases, you may find that you need to regenerate the certificates and security credentials (private and public keys) generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master that you need...
to move to a different network in your infrastructure, or you may find that you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

Related information

Regenerating certificates: split installs on page 710
In some cases, you may find that you need to regenerate the SSL certificates and security credentials (private and public keys) that are generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master you need to move to a different network in your infrastructure, or you may find you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

Regenerate Puppet agent certificates on page 717
From time to time, you might encounter a situation in which you need to regenerate a certificate for an agent. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you're receiving SSL errors on your agent that are preventing you from performing normal operations.

Generate a custom Diffie-Hellman parameter file on page 723
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.

Regenerate certificates in PE: monolithic installs
You can regenerate all certificates in a monolithic PE deployment, including the certificates and keys for the Puppet master, PuppetDB, console, and associated services.

Before you begin

- You must be logged in as a root to make these changes.
- In the following instructions, when `<CERTNAME>` is used, it refers to the agent's certname. To find this value, run `puppet config print certname` before starting.

Regenerating your certificates will invalidate all existing authentication tokens. Once the regeneration process is complete, all PE users must generate new authentication tokens.

Back up certificate directories
If something goes wrong during the regeneration process, you may need to restore these directories so your deployment can stay functional. However, if you needed to regenerate your certs for security reasons and couldn't, you should contact Puppet support as soon as you restore service so we can help you secure your site.

Back up the following directories:

- `/etc/puppetlabs/puppet/ssl/`
- `/etc/puppetlabs/puppet/ssl/`
- `/etc/puppetlabs/puppetdb/ssl/`
- `/opt/puppetlabs/server/data/console-services/certs/`
- `/opt/puppetlabs/server/data/postgresql/9.6/data/certs/`
- `/etc/puppetlabs/orchestration-services/ssl`
(Optional) Delete and recreate the master CA
If needed, you can delete and recreate the Puppet CA before regenerating the rest of your monolithic certificates.

As an alternative to performing these steps manually, on your master logged in as root, run `puppet infrastructure run rebuild_certificate_authority caserver=<CA_SERVER_HOSTNAME>`. If your master operates as your CA server, specify `caserver=localhost`. (If you're running PE version 2018.1.11 or newer, omit the `caserver` parameter.) Running the command with `localhost` avoids the requirement to set up SSH between your master and itself.

The `puppet infrastructure run` command leverages built-in plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your master 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 master. For more information, see [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`.

---

**CAUTION:** This is an optional step and is meant for use in the event of a total compromise of your site, or some other unusual circumstance. This destroys the certificate authority and all other certificates.

Run the following commands on your master or CA server.

1. Delete the CA and clear all certs from your master: `rm -rf /etc/puppetlabs/puppet/ssl/*`
2. Regenerate the CA: `puppet cert list -a`

You should see this message: **Notice: Signed certificate request for ca**
Regenerate the Puppet master certificates

In this step, you'll create the certificates for the Puppet master and then configure PE so the certificate is available to PE's components and services.

As an alternative to performing these steps manually, on your master logged in as root, run puppet infrastructure run regenerate_master_certificate.

You can specify these optional parameters:

- *tmpdir* — Path to a directory to use for uploading and executing temporary files.
- *dns_alt_names* – Comma-separated list of alternate DNS names to be added to the certificates generated for your master.

**Important:** To use the *dns_alt_names* parameter, you must configure Puppet Server with *allow-subject-alt-names* in the certificate-authority section of *pe-puppet-server.conf*. 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 alt names included in the entry when regenerating certificates.

The *puppet infrastructure run* command leverages built-in plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your master 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 master. For more information, see [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.

1. Remove the Puppet master's cached catalog: `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`

2. Clear the cert for the Puppet master: `puppet cert clean <CERTNAME>`

   **Note:** This step is not necessary if you deleted and recreated the CA cert.

3. Generate the certificates for PE services and update the configuration of PE: `puppet infrastructure configure --no-recover`

   **Note:** Be sure to specify any DNS alt names you have in the `pe_install::puppet_master_dnsaltnames` array in `/etc/puppetlabs/enterprise/conf.d/pe.conf`. You can find the list of your current DNS alt names with `puppet cert list <CERTNAME>`. By default, PE uses `puppet` and `puppet.domain`.

4. Run Puppet on the Puppet master: `puppet agent -t`

   A successful Puppet run is necessary to ensure that PE's services are properly configured.
Related information

Regenerate compile master certs on page 718
From time to time, you may encounter a situation in which you need to regenerate a certificate for a compile master. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you're receiving strange SSL errors on your compile master that are preventing you from performing normal operations.

Regenerating certificates: split installs

In some cases, you may find that you need to regenerate the SSL certificates and security credentials (private and public keys) that are generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master that you need to move to a different network in your infrastructure, or you may find that you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

Related information

Regenerating certificates: monolithic installs on page 706
In some cases, you may find that you need to regenerate the certificates and security credentials (private and public keys) generated by PE's built-in certificate authority (CA). For example, you may have a Puppet master that you need to move to a different network in your infrastructure, or you may find that you need to regenerate all the certificates and security credentials in your infrastructure due to an unforeseen security vulnerability.

Regenerate Puppet agent certificates on page 717
From time to time, you might encounter a situation in which you need to regenerate a certificate for an agent. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you're receiving SSL errors on your agent that are preventing you from performing normal operations.

Generate a custom Diffie-Hellman parameter file on page 723
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.

Regenerate certificates in PE: split installs

You can regenerate all certificates in a split PE deployment including the certificates and keys for the Puppet master, PuppetDB, console, and associated services.

Before you begin

- You must be logged in as a root to make these changes.
- In the following instructions, when `<CERTNAME>` is used, it refers to the agent's certname on each node. To find this value, run `puppet config print certname` before starting.

Regenerating your certificates will invalidate all existing authentication tokens. Once the regeneration process is complete, all PE users must generate new authentication tokens.

Regenerating your certificates involves the following tasks:

1. Back up certificate directories
2. (Optional) Delete and recreate the Puppet certificate authority (CA)
3. Regenerate the Puppet master, console, and PuppetDB certificates
4. Configure PE

Back up certificate directories
If something goes wrong during the regeneration process, you may need to restore these directories so your deployment can stay functional. However, if you needed to regenerate your certs for security reasons and couldn't, you should contact Puppet support as soon as you restore service so we can help you secure your site.
1. On the Puppet master, back up the following directories:
   - /etc/puppetlabs/puppet/ssl/
   - /etc/puppetlabs/orchestration-services/ssl

2. On the PuppetDB node, back up the following directories:
   - /etc/puppetlabs/puppet/ssl/
   - /etc/puppetlabs/puppetdb/ssl/
   - /opt/puppetlabs/server/data/postgresql/9.6/data/certs/

3. On the console, back up the following directories:
   - /etc/puppetlabs/puppet/ssl/
   - /opt/puppetlabs/server/data/console-services/certs/

(Optional) Delete and recreate the master CA
If needed, you can delete and recreate the Puppet CA before regenerating the rest of your monolithic certificates.

⚠️ CAUTION: This is an optional step and is meant for use in the event of a total compromise of your site, or some other unusual circumstance. This destroys the certificate authority and all other certificates.

Run the following commands on your master or CA server.
1. Delete the CA and clear all certs from your master: `rm -rf /etc/puppetlabs/puppet/ssl/*`
2. Regenerate the CA: `puppet cert list -a`
   
   You should see this message: Notice: Signed certificate request for ca

Regenerate the Puppet master certificates
In this step, you'll create the certificates for the split Puppet master.

Run the following commands on the Puppet master.
1. Remove the Puppet master's cached catalog: `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`
2. Clear the cert for the Puppet master: `puppet cert clean <CERTNAME>`
   
   Note: This step is not necessary if you deleted and recreated the CA cert.

Clear the PuppetDB certificates
In this step, you clear the PuppetDB certificate.

2. Clear the cert for the PuppetDB node. On the Puppet master, run: `puppet cert clean <CERTNAME>`
3. Remove the certificates. On the PuppetDB node, run: `rm -f /etc/puppetlabs/puppet/ssl/*/<CERTNAME>.pem`

Clear the PE console certificates
In this step, you clear the console certificate.

1. Remove the console's cached catalog. On the console node, run: `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`
2. Clear the cert for the console node. On the Puppet master, run: `puppet cert clean <CERTNAME>`
3. Remove the certificates. On the console node, run: `rm -f /etc/puppetlabs/puppet/ssl/*/<CERTNAME>.pem`
Update the configuration of PE
In this step, you configure PE to generate new certificates for the component nodes and update PE's configuration.

1. On the Puppet master node, run: `puppet infrastructure configure --no-recover`

   **Note:** Be sure to specify any DNS alt names you have in the `pe_install::puppet_master_dnsaltnames` array in `/etc/puppetlabs/enterprise/conf.d/pe.conf`. You can find the list of your current DNS alt names with `puppet cert list <CERTNAME>`. By default, PE uses `puppet` and `puppet.domain`.

2. On the PuppetDB node, run: `puppet infrastructure configure --no-recover`

3. On the console node, run: `puppet infrastructure configure --no-recover`

4. Run Puppet on each node in the following order:
   a) Puppet master
   b) PuppetDB
   c) Console

   A successful Puppet run on each node, in the given order, is necessary to ensure that PE's services are properly configured.

Individual PE component cert regeneration (split installs only)

If you encounter a security vulnerability, or need to change your certificates for some other reason (for example, if you have a hostname change), you can regenerate the certificate and security credentials (private and public keys) generated for the PE components.

**CAUTION:** If you've experienced an unforeseen security vulnerability and need to regenerate all the certificates and security credentials in your infrastructure, refer to Regenerating certs and security credentials in split Puppet Enterprise installations for complete instructions.

On monolithic installs, the Puppet master shares an agent cert and security credentials with the PuppetDB and the console. For a monolithic install, you must regenerate all certs and security credentials, as documented in Regenerating certificates: monolithic installs.

You can regenerate certificates for the following components:

- The Puppet master
- PuppetDB
- The PE console

**Related information**

Regenerate certificates in PE: monolithic installs on page 707

You can regenerate all certificates in a monolithic PE deployment, including the certificates and keys for the Puppet master, PuppetDB, console, and associated services.

Regenerate certificates in PE: split installs on page 710

You can regenerate all certificates in a split PE deployment including the certificates and keys for the Puppet master, PuppetDB, console, and associated services.

Regenerate Puppet master certs (split installs)

You can regenerate all certificates for the Puppet master server only, including the certificates and keys for associated services running on the Puppet master.

**Before you begin**

- You must be logged in as a root, (or in the case of Windows agents, as an account with Administrator Privileges) to make these changes.
- In the following instructions, when `<CERTNAME>` is used, it refers to the Puppet master's certname. To find this value, run `puppet config print certname` before starting.
CAUTION: If you’ve experienced an unforeseen security vulnerability and need to regenerate all the certificates and security credentials in your infrastructure, refer to Regenerating certs and security credentials in split Puppet Enterprise installations for complete instructions.

On monolithic installs, the Puppet master shares an agent cert and security credentials with the PuppetDB and the console. For a monolithic install, you must regenerate all certs and security credentials, as documented in Regenerating certificates: monolithic installs.

This document should not be used to regenerate certificates for compile masters. Instead, refer to the compile master cert regen instructions.

If you encounter any errors during steps that involve service stop/start, rm, cp, or chmod commands, diagnose these before continuing, as the success of each step is important to the success of the next step.

Unless otherwise indicated, all commands are run on the Puppet master server.

1. Back up the /etc/puppetlabs/puppet/ssl/ directory.
   If something goes wrong, you may need to restore these directories so your deployment can stay functional. However, if you needed to regenerate your certs for security reasons and couldn’t, you should contact Puppet support as soon as you restore service so we can help you secure your site.

2. Shut down all PE-related services with the following commands:
   puppet resource service puppet ensure=stopped
   puppet resource service pe-puppetserver ensure=stopped
   puppet resource service pe-activemq ensure=stopped
   puppet resource service mcollective ensure=stopped
   puppet resource service pe-orchestration-services ensure=stopped
   puppet resource service pxp-agent ensure=stopped

3. Clear the cert and security credentials for the Puppet master: puppet cert clean <CERTNAME>

4. Remove the cached catalog: rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json

5. Generate the Puppet master’s new certs: puppet cert generate <CERTNAME> --dns_alt_names=<DNS_ALT_NAMES>
   Note: Be sure to specify any DNS alt names you have. You can find the list of your current DNS alt names with puppet cert list <CERTNAME>. By default, PE uses puppet and puppet.domain.

6. Copy the new cert and security credentials to the orchestration-services cert directory.
   cp /etc/puppetlabs/puppet/ssl/certs/<CERTNAME>.pem /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.cert.pem
   cp /etc/puppetlabs/puppet/ssl/public_keys/<CERTNAME>.pem /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.public_key.pem
   cp /etc/puppetlabs/puppet/ssl/private_keys/<CERTNAME>.pem /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.private_key.pem
   openssl pkcs8 -topk8 -inform PEM -outform DER -in /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.private_key.pem -out /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.private_key.pk8 -nocrypt
   chown -R pe-orchestration-services:pe-orchestration-services /etc/puppetlabs/orchestration-services/ssl/

7. Create the orchestration-services .pk8 cert.
   cd /etc/puppetlabs/orchestration-services/ssl
   openssl pkcs8 -topk8 -inform PEM -outform DER -in /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.private_key.pem -out /etc/puppetlabs/orchestration-services/ssl/<CERTNAME>.private_key.pk8 -nocrypt
   chown -R pe-orchestration-services:pe-orchestration-services /etc/puppetlabs/orchestration-services/ssl/
8. Restart all PE-related services with the following commands:

```
puppet resource service puppet ensure=running
puppet resource service pe-puppetserver ensure=running
puppet resource service pe-activemq ensure=running
puppet resource service mcollective ensure=running
puppet resource service pe-orchestration-services ensure=running
puppet resource service pxp-agent ensure=running
```

Regenerate PuppetDB certs (split installs)

You can regenerate all certificates for the PuppetDB only, including the certificates and keys for associated services running on PuppetDB.

**Before you begin**

- You must be logged in as a root, (or in the case of Windows agents, as an account with Administrator Privileges) to make these changes.
- In the following instructions, when `<CERTNAME>` is used, it refers to the Puppet master's certname. To find this value, run `puppet config print certname` before starting.

**CAUTION:** If you've experienced an unforeseen security vulnerability and need to regenerate all the certificates and security credentials in your infrastructure, refer to Regenerating certs and security credentials in split Puppet Enterprise installations for complete instructions.

On monolithic installs, the Puppet master shares an agent cert and security credentials with the PuppetDB and the console. For a monolithic install, you must regenerate all certs and security credentials, as documented in Regenerating certificates: monolithic installs.

If you encounter any errors during steps that involve `service stop/start`, `rm`, `cp`, or `chmod` commands, diagnose these before continuing, as the success of each step is important to the success of the next step.

Unless otherwise indicated, all commands are run on the PuppetDB server.

1. On the PuppetDB node, back up the following directories:
   - `/etc/puppetlabs/puppet/ssl/`
   - `/etc/puppetlabs/puppetdb/ssl/`
   - `/opt/puppetlabs/server/data/postgresql/9.6/data/certs/*`

2. On the PuppetDB node, shut down all PE-related services with the following commands:

```
puppet resource service puppet ensure=stopped
puppet resource service pe-puppetdb ensure=stopped
puppet resource service pe-postgresql ensure=stopped
puppet resource service mcollective ensure=stopped
```

3. On the PuppetDB node, delete the agent's SSL cert and security credentials: `rm -rf /etc/puppetlabs/puppet/ssl/*`

4. On the master or CA server, remove the cert for the PuppetDB node: `puppet cert clean <PUPPETDB CERTNAME>`

5. On the master, remove the cached catalog: `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`

6. On the PuppetDB node, generate security credentials and request a new certificate from the CA master: `puppet agent --test --no-daemonize --noop`

   These certs will end up in `/etc/puppetlabs/puppet/puppet/ssl`.

**Note:** This agent run will not complete successfully, but it is necessary to set up the agent certificate for the PuppetDB node. You will see some errors about node definition and the inability to submit facts due to PuppetDB being offline. You can ignore these.
7. On the PuppetDB node, delete PuppetDB’s SSL cert and security credentials:
   ```bash
   rm -rf /etc/puppetlabs/puppetdb/ssl/*
   ```

8. On the PuppetDB node, copy the agent’s certs and security credentials to the PuppetDB SSL directory:
   ```bash
   cp /etc/puppetlabs/puppet/ssl/certs/<CERTNAME>.pem /etc/puppetlabs/puppetdb/ssl/<CERTNAME>.cert.pem
   cp /etc/puppetlabs/puppet/ssl/public_keys/<CERTNAME>.pem /etc/puppetlabs/puppetdb/ssl/<CERTNAME>.public_key.pem
   cp /etc/puppetlabs/puppet/ssl/private_keys/<CERTNAME>.pem /etc/puppetlabs/puppetdb/ssl/<CERTNAME>.private_key.pem
   ```

9. On the PuppetDB node, create PuppetDB’s .pk8 cert.
   ```bash
   cd /etc/puppetlabs/puppetdb/ssl
   openssl pkcs8 -topk8 -inform PEM -outform DER -in /etc/puppetlabs/puppetdb/ssl/<CERTNAME>.private_key.pem -out /etc/puppetlabs/puppetdb/ssl/<CERTNAME>.private_key.pk8 -nocrypt
   chown -R pe-puppetdb:pe-puppetdb /etc/puppetlabs/puppetdb/ssl
   ```

10. On the PuppetDB node, clear the certs and security credentials from the PostgreSQL certs directory:
    ```bash
    rm -rf /opt/puppetlabs/server/data/postgresql/9.6/data/certs/*
    ```

11. On the PuppetDB node, copy the certs and security credentials to the PostgreSQL certs directory:
    ```bash
    cp /etc/puppetlabs/puppet/ssl/certs/<CERTNAME>.pem /opt/puppetlabs/server/data/postgresql/9.6/data/certs/_local.cert.pem
    cp /etc/puppetlabs/puppet/ssl/private_keys/<CERTNAME>.pem /opt/puppetlabs/server/data/postgresql/9.6/data/certs/_local.private_key.pem
    chmod 400 /opt/puppetlabs/server/data/postgresql/9.6/data/certs/*
    chown pe-postgres:pe-postgres /opt/puppetlabs/server/data/postgresql/9.6/data/certs/*
    ```

12. On the PuppetDB node, restart all PE-related services with the following commands:
    ```bash
    puppet resource service puppet ensure=running
    puppet resource service pe-puppetdb ensure=running
    puppet resource service pe-postgresql ensure=running
    puppet resource service mcollective ensure=running
    ```

### Regenerate PE console certs

You can regenerate all certificates for the console only, including the certificates and keys for associated services running on the console.

**Before you begin**

- You must be logged in as a root, (or in the case of Windows agents, as an account with Administrator Privileges) to make these changes.
- In the following instructions, when `<CERTNAME>` is used, it refers to the Puppet master's certname. To find this value, run `puppet config print certname` before starting.

**CAUTION:** If you've experienced an unforeseen security vulnerability and need to regenerate all the certificates and security credentials in your infrastructure, refer to Regenerating certs and security credentials in split Puppet Enterprise installations for complete instructions.

On monolithic installs, the Puppet master shares an agent cert and security credentials with the PuppetDB and the console. For a monolithic install, you must regenerate all certs and security credentials, as documented in Regenerating certificates: monolithic installs.

If you encounter any errors during steps that involve `service stop/start, rm, cp, or chmod` commands, diagnose these before continuing, as the success of each step is important to the success of the next step.

© 2020 Puppet, Inc.
Unless otherwise indicated, all commands are run on the console server.

1. On the console node, back up the following directories:

   - `/etc/puppetlabs/puppet/ssl/`
   - `/opt/puppetlabs/server/data/console-services/certs`

2. On the console node, shut down all PE-related services with the following commands:

   ```
   puppet resource service puppet ensure=stopped
   puppet resource service pe-console-services ensure=stopped
   puppet resource service pe-nginx ensure=stopped
   puppet resource service mcollective ensure=stopped
   ```

3. On the console node, delete the Puppet agent's SSL cert and security credentials:

   ```
   rm -rf /etc/puppetlabs/puppet/ssl/*
   ```

4. On the Puppet master, or CA server, remove the cert for the console node:

   ```
   puppet cert clean <CERTNAME>
   ```

5. On the Puppet master, remove the cached catalog:

   ```
   rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json
   ```

6. On the console node, generate security credentials and request a new certificate from the CA Puppet master:

   ```
   puppet agent --test --no-daemonize --noop
   ```

   These certs will end up in `/etc/puppetlabs/puppet/ssl`.

   **Note:** This agent run will not complete successfully, but it is necessary to set up the agent certificate for the node. You will see some errors about node definition and the inability to execute http requests due to the console being offline. You can ignore these.

7. On the console node, purge the console-services directory:

   ```
   rm -rf /opt/puppetlabs/server/data/console-services/certs/*
   ```

8. On the console node, copy the agent's cert and security credentials to the console-services cert directory.

   ```
   cp /etc/puppetlabs/puppet/ssl/certs/<CERTNAME>.pem /opt/puppetlabs/server/data/console-services/certs/<CERTNAME>.cert.pem
   cp /etc/puppetlabs/puppet/ssl/public_keys/<CERTNAME>.pem /opt/puppetlabs/server/data/console-services/certs/<CERTNAME>.public_key.pem
   cp /etc/puppetlabs/puppet/ssl/private_keys/<CERTNAME>.pem /opt/puppetlabs/server/data/console-services/certs/<CERTNAME>.private_key.pem
   ```

9. Create the console-services .pk8 cert.

   ```
   cd /opt/puppetlabs/server/data/console-services/certs/
   openssl pkcs8 -topk8 -inform PEM -outform DER -in /opt/puppetlabs/server/data/console-services/certs/<CERTNAME>.private_key.pem -out /opt/puppetlabs/server/data/console-services/certs/<CERTNAME>.private_key.pk8 -nocrypt
   chown -R pe-console-services:pe-console-services /opt/puppetlabs/server/data/console-services/certs/
   ```

10. On the console node, ensure the console can access the new credentials.

    ```
    chown -R pe-console-services:pe-console-services /opt/puppetlabs/server/data/console-services/certs
    ```

11. On the console node, restart all PE-related services with the following commands:

    ```
    puppet resource service puppet ensure=running
    puppet resource service pe-nginx ensure=running
    puppet resource service pe-console-services ensure=running
    puppet resource service mcollective ensure=running
    ```
Regenerate Puppet agent certificates

From time to time, you might encounter a situation in which you need to regenerate a certificate for an agent. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you’re receiving SSL errors on your agent that are preventing you from performing normal operations.

As an alternative to performing these steps manually, on your master logged in as root, run puppet infrastructure run regenerate_agent_certificate agent=<AGENT_HOSTNAME> caserver=<CA_SERVER_HOSTNAME>. Your CA server is usually your master. (If you’re running PE version 2018.1.11 or newer, omit the caserver parameter.)

To regenerate certificates for multiple agents, use a comma-separated list of the agents’ hostnames. For example, puppet infrastructure run regenerate_agent_certificate agent=agent1.example.net,agent2.example.net,agent3.example.net

You can specify these optional parameters:

- **tmpdir** — Path to a directory to use for uploading and executing temporary files.
- **dns_alt_names** — Comma-separated list of alternate DNS names to be added to the certificates generated for your agents.

Important: To use the dns_alt_names parameter, you must configure Puppet Server with allow-subject-alt-names in the certificate-authority section of pe-puppet-server.conf. 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 alt names included in the entry when regenerating certificates.

The puppet infrastructure run command leverages built-in plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your master 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 master. For more information, see 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.

Unless otherwise indicated, the following steps should be performed on your agent node.

1. On the master, clear the cert for the agent node. Run puppet cert clean <CERTNAME>.
2. On the agent, back up the `/etc/puppetlabs/puppet/ssl/` directory.
   If something goes wrong, you may need to restore these directories so your deployment can stay functional. However, if you needed to regenerate your certs for security reasons and couldn’t, you should contact Puppet support as soon as you restore service so we can help you secure your site.

3. Stop the Puppet agent, MCollective, and PXP agent services.

   ```bash
   puppet resource service puppet ensure=stopped
   puppet resource service mcollective ensure=stopped
   puppet resource service pxp-agent ensure=stopped
   ```

4. Delete the agent's SSL directory.
   - On *nix nodes, run `rm -rf /etc/puppetlabs/puppet/ssl`
   - On Windows nodes, delete the `$confdir\ssl` directory, using the Administrator confdir.

5. Remove the agent's cached catalog.
   - On *nix nodes, run `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`
   - On Windows nodes, delete the `$client_datadir\catalog\<CERTNAME>.json` file, using the Administrator confdir.

6. Re-start the Puppet agent and MCollective services.

   ```bash
   puppet resource service puppet ensure=running
   puppet resource service mcollective ensure=running
   ```

   After the agent starts, it will automatically generate keys and request a new certificate from the CA Puppet master.

7. If you are not using autosigning, you will need to sign each agent node’s certificate request. You can do this with the console's request manager, or by logging into the CA Puppet master server, running

   ```bash
   puppet cert list
   puppet cert sign <NAME>
   ```

   After the agent node’s certificate is signed, you can either manually kick off a Puppet run from the console or command line, or wait for the agent to run based on the runinterval, the default of which is 30 minutes. At this point, the agent will perform a full catalog run, restart the PXP agent service, and will resume its role in your PE deployment.

   **Tip:** For information about autosigning, see [Autosigning certificate requests](#).

---

### Regenerate compile master certs

From time to time, you may encounter a situation in which you need to regenerate a certificate for a compile master. Perhaps there is a security vulnerability in your infrastructure that you can remediate with a certificate regeneration, or maybe you’re receiving strange SSL errors on your compile master that are preventing you from performing normal operations.

Unless otherwise indicated, the following steps are performed on your compile master nodes.

1. Log into the master of masters (MoM) as root.
2. On the MoM, run `puppet cert clean <COMPILE MASTER HOSTNAME>`.
3. Log into the compile master node as root.
4. Back up the `/etc/puppetlabs/puppet/ssl/` directory. Run `cp -r /etc/puppetlabs/puppet/ssl/ /etc/puppetlabs/puppet/ssl_bak/`.
   If something goes wrong, you can restore this directory to keep your deployment functioning.

© 2020 Puppet, Inc.
5. Stop the Puppet agent, MCollective, and PXP agent services.

   puppet resource service puppet ensure=stopped
   puppet resource service pe-puppetserver ensure=stopped
   puppet resource service mcollective ensure=stopped
   puppet resource service pxp-agent ensure=stopped

6. Delete the compile master's SSL directory. Run `rm -rf /etc/puppetlabs/puppet/ssl`.


8. Re-start the Puppet agent service and manually trigger a Puppet run, or wait for the next automatically scheduled run.

   puppet resource service puppet ensure=running

9. Log into the MoM as root.


11. Log into the compile master as root.

12. Run Puppet.

   PE performs a full catalog run, and the compile master will resume its role in your PE deployment.

---

**Use the PE CA as an intermediate CA**

PE can operate as an intermediate CA to an external root CA. (It can't be an intermediate to an intermediate.) In this configuration, the PE CA is left enabled and it can automatically accept CSRs, distribute certificates to agents, and use the standard `puppet cert` command to sign certificates.

**Before you begin**

1. You must have an intermediate CA certificate from your external root CA.
2. Back up your infrastructure.
3. Stop all Puppet services.

   puppet resource service puppet ensure=stopped
   puppet resource service pe-puppetserver ensure=stopped
   puppet resource service pe-activemq ensure=stopped
   puppet resource service mcollective ensure=stopped
   puppet resource service pe-puppetdb ensure=stopped
   puppet resource service pe-postgresql ensure=stopped
   puppet resource service pe-console-services ensure=stopped
   puppet resource service pe-nginx ensure=stopped
   puppet resource service pe-orchestration-services ensure=stopped
   puppet resource service pxp-agent ensure=stopped

4. On the master, remove existing `.pem` files: `find /etc/puppetlabs/puppet/ssl -name '*.pem' -delete`

There are some limitations to this configuration:

- Agent-side CRL checking is not possible, although Puppet Server still verifies the CRL.
- The CA certificate bundle (the external root CA combined with the intermediate CA certificate) must be distributed to the agents manually, ideally before Puppet runs.

**CAUTION:** This procedure destroys all certificates currently in use in your infrastructure and prevents normal operation until certificates are restored. Time performing this task accordingly.
**Configure PE to use the new certificates**

Place and configure certificates on your Puppet Server.

Perform these steps on your Puppet Server.

1. Put these files in the locations indicated:

<table>
<thead>
<tr>
<th>File</th>
<th>Location</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>intermediate CA certificate</td>
<td>/etc/puppetlabs/puppet/ssl/ca/ca_crt.pem</td>
<td>Can't have a passphrase, because the PE CA can't provide one when using the key.</td>
</tr>
<tr>
<td>intermediate CA key</td>
<td>/etc/puppetlabs/puppet/ssl/ca/ca_key.pem</td>
<td>Can be placed anywhere, because Puppet Server doesn't use it directly, but the rest of these instructions assume you placed it as shown.</td>
</tr>
<tr>
<td>root CA certificate</td>
<td>/etc/puppetlabs/puppet/ssl/ca/root_crt.pem</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** All certificate files must have their ownership set to `pe-puppet:pe-puppet` and have permissions of `0600`.

2. Generate the CA bundle, by combining the root CA and intermediate CA certificates into one PEM file.

   ```
   cd /etc/puppetlabs/puppet/ssl/ca
   cat ca_crt.pem root_crt.pem > ../certs/ca.pem
   ```

3. Update the CA serial file:

   ```
   printf '%04x' 3 > /etc/puppetlabs/puppet/ssl/ca/serial
   ```

4. Generate a certificate for Puppet Server to use, including any `dns_alt_names` that the server must service.

   ```
   puppet cert generate puppetserver.my.domain.net --dns_alt_names=puppetserver,puppet
   ```

5. Configure a CRL file from the root CA.

   - If you have a pre-generated CRL from the root CA, install it into `/etc/puppetlabs/puppet/ssl/ca/ca_crl.pem`.
   - If you don't have a pre-generated CRL from the root CA, create one by running `puppet cert generate fakehost` and then revoking this certificate with `puppet cert clean fakehost`.

6. Re-generate the certificate inventory file.

   ```
   puppet cert reinventory
   ```

**Reconfigure PE infrastructure**

After replacing the public key infrastructure, you must regenerate all required certificates.

Perform these steps on your master.

1. Set `certificate revocation` to `false` in the main section of `puppet.conf`.

   ```
   [main]
   certificate_revocation = false
   ```

2. Remove the cached catalog:

   ```
   rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/ <PUPPET MASTER FQDN>.json
   ```

3. Reconfigure PE:

   ```
   puppet infrastructure configure
   ```

4. Run Puppet:

   ```
   puppet agent -t
   ```
Update agents to use the new CA

Prepare agents to use the new CA configuration by copying the CA bundle in place and configuring certificate revocation.

Before you begin
Back up the /etc/puppetlabs/puppet/ssl/ directory. If something goes wrong, you might need to restore these directories so your deployment remains functional.

Tip: If you were regenerating certs for security reasons and the process failed, contact Support as soon as you restore service so we can help you secure your site.

Complete these steps on each agent, unless otherwise indicated.

1. Stop Puppet services:

   puppet resource service puppet ensure=stopped
   puppet resource service mcollective ensure=stopped
   puppet resource service pxp-agent ensure=stopped
   # Stop pe-puppetserver if the agent is a compile master
   puppet resource service pe-puppetserver ensure=stopped

2. Copy the CA bundle you created to /etc/puppetlabs/puppet/ssl/certs/ca.pem.
   
   Note: If you copy this file into place before the first Puppet run, you won't receive any errors. If you attempt a Puppet run prior to this file being present, you receive errors because the auto-distributed ca.pem file doesn't include the root CA.

3. On your master, set certificate revocation to false in the main section of puppet.conf:

   [main]
   certificate_revocation = false

4. Remove the certificate file, certificate revocation list (CRL), and the agent's cached catalog:

   find /etc/puppetlabs/puppet/ssl -name "$(puppet config print certname).pem" -or -name crl.pem -delete
   rm -f "/opt/puppetlabs/puppet/cache/client_data/catalog/$(puppet config print certname).json"

5. Restart the agent service: puppet resource service puppet ensure=running

6. If you aren't using autosigning, from the console or your master's command line, sign the agent's certificate signing request.

7. From the console or your master's command line, run Puppet: puppet agent -t

   The agent performs a full catalog run, restarts the PXP agent service, and resumes its role in your deployment.

Use a custom SSL certificate for the console

The 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. You might find that this is not an acceptable scenario and want to use a custom CA to create the console's certificate.

Before you begin
• You should have 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 needs to contain a complete chain, including the applicable root CA.
• The keys and certs used in this procedure must be in PEM format.
1. Retrieve the custom certificate's public and private keys, and, for ease of use, name them as follows:
   • public-console.cert.pem
   • public-console.private_key.pem
2. Add the files from step 1 to /opt/puppetlabs/server/data/console-services/certs/.
   If you have a split install, this directory is on the console node.
3. Use the console to edit the parameters of the puppetEnterprise::profile::console class.
   a) Click Classification, and in the PE Infrastructure group, select the PE Console group.
   b) On the Configuration tab, in the puppetEnterprise::profile::console class, add the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>browser_ssl_cert</td>
<td>/opt/puppetlabs/server/data/console-services/certs/public-console.cert.pem</td>
</tr>
<tr>
<td>browser_ssl_private_key</td>
<td>/opt/puppetlabs/server/data/console-services/certs/public-console.private_key.pem</td>
</tr>
</tbody>
</table>
   c) Commit changes.
4. Run Puppet.
5. When the run is complete, restart the console and the nginx service for the changes to take effect.

   puppet resource service pe-console-services ensure=stopped
   puppet resource service pe-console-services ensure=running
   puppet resource service pe-nginx ensure=stopped
   puppet resource service pe-nginx ensure=running

   You should now be able to navigate to your console and see the custom certificate in your browser.

Change the hostname of a monolithic master

To change the hostnames assigned to your PE infrastructure nodes requires updating the corresponding PE certificate names. You might have to update your master hostname, for example, when migrating to a new PE installation, or after an organizational change such as a change in company name.

**Before you begin**

Download and install the puppetlabs/support_tasks module.

Make sure you are using the latest version of the support_tasks module.

To update the hostname of your master:

1. On the master, set the new hostname by running the following command:

   hostnamectl set-hostname newhostname.example.com

2. Make sure the hostname -f command returns the new fully qualified hostname, and that it resolves to the same IP address as the old hostname, by adding an entry for the new hostname in /etc/hosts:

   <IP address> <newhostname.example.com> <oldhostname.example.com> ...

© 2020 Puppet, Inc.
3. Run a task for changing the host name against the old certificate name on the master, using one of the following methods:

   • Using Bolt:

     bolt task run support_tasks::st0263_rename_pe_master -n $(puppet config print certname) --modulepath="/etc/puppetlabs/code/environments/production/modules"

     **Note:** When running the task, Bolt must be using the default SSH transport, rather than the PCP protocol, to avoid errors when services are restarted.

   • Using the command line:

     puppet task run support_tasks::st0263_rename_pe_master -n $(puppet config print certname)

     The task restarts all Puppet services, which causes a connection error. You can ignore the error while the task continues to run in the background. To check if the task is complete, tail /var/log/messages. When the output from the puppet agent -t command is displayed in the system log, the task is complete. For example:

     ```
     # tail /var/log/messages
     Aug 15 09:08:29 oldhostname systemd: Reloaded pe-orchestration-services Service.
     Aug 15 09:08:29 oldhostname puppet-agent[4780]: (/Stage[main]/Puppet_enterprise::Profile::Orchestrator/Puppet Enterprise::Trapperkeeper::Pe_service[pe-orchestration-services]/Service[pe-orchestration-services]) Triggered 'refresh' from 1 event
     Aug 15 09:08:34 oldhostname puppet-agent[4780]: Applied catalog in 19.16 seconds
     ```

### 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 the console node, (for a mono install, this is the same node as a the Puppet master), run the following command:

   ```
   /opt/puppetlabs/puppet/bin/openssl dhparam -out /etc/puppetlabs/nginx/<PROXY-CUSTOM-dhparam>.pem 2048
   ```

   **Note:** After running this command, PE may take several minutes to complete this step.

2. On the Puppet master, open your `pe.conf` file (located at `/etc/puppetlabs/enterprise/conf.d/pe.conf`) and add the following parameter and value:

   ```
   "puppet_enterprise::profile::console::proxy::dhparam_file": "/etc/puppetlabs/nginx/<PROXY-CUSTOM-dhparam>.pem"
   ```

3. On the console node, run Puppet: `puppet agent -t`. 

© 2020 Puppet, Inc.
Disable TLSv1 in PE

You can disable TLSv1 in PE to comply with standards as necessary.

The services running in PE support versions 1, 1.1, and 1.2 of the Transport layer security (TLS) protocol but use TLSv1 by default. The Payment Card Industry Data Security Standard (PCI DSS) requires TLSv1 to be permanently disabled by 30 June, 2018. To comply with PCI DSS, or simply to tighten your own security, disable TLSv1.

PE uses TLSv1 by default because the PXP agent service running on older agents use TLSv1. In PE you can disable TLSv1, but the first step is upgrading your agents to 2017.2 or later.

**Note:** AIX supports only TLSv1. If you disable TLSv1, install AIX agents with your own package management instead of PE package management.

1. Upgrade your *nix or Windows agents to the latest version of PE (must be 2017.2 or later).
2. In the console, click **Classification > PE Infrastructure**
3. On the **Configuration** tab, add the following parameter and value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet_enterprise::ssl_protocols</td>
<td>[&quot;TLSv1.1&quot;, &quot;TLSv1.2&quot;]</td>
</tr>
</tbody>
</table>

4. Click **Add parameter**, and commit changes.
5. In a monolithic installation, run Puppet on the Puppet master. In a split installation, run Puppet on the Puppet master, console, and PuppetDB nodes.

Managing MCollective

The MCollective engine can invoke many kinds of actions in parallel across any number of nodes.

**CAUTION:**

2018.1 is the last release to support Marionette Collective, also known as MCollective. While 2018.1 remains supported, will continue to address security issues for MCollective. Feature development has been discontinued. Future releases of will not include MCollective. For more information, see the Puppet Enterprise support lifecycle.

To prepare for these changes, migrate your MCollective work to orchestrator to automate tasks and create consistent, repeatable administrative processes. Use orchestrator to automate your workflows and take advantage of its integration with console and commands, APIs, role-based access control, and event tracking.
With MCollective in Puppet Enterprise, you can invoke many kinds of actions in parallel across any number of nodes. Several useful actions are available by default, and you can easily add and use new actions.

**Invoking actions** on page 727
You can invoke MCollective actions on both *nix and Windows machines, based on commands you run from the command line on a Linux-based Puppet master servers.

**Controlling Puppet** on page 732
PE's configuration management features rely on the Puppet agent service, which runs on every node and fetches configurations from the Puppet master server.

**List of built-in actions** on page 735
MCollective comes with several built-in actions. You invoke these actions from the command line.

**Adding actions and plugins to PE** on page 745
You can extend PE's MCollective engine by adding new actions. Actions are distributed in agent plugins, which are bundles of several related actions. You can write your own agent plugins (or download ones created by other people), and use PE to install and configure them on your nodes.

**Disabling MCollective** on page 749
Turn off MCollective on a single node or on all nodes.

**Change the port used by MCollective/ActiveMQ** on page 750
You can change the port that MCollective/ActiveMQ uses with a simple variable change in the console.

**Moving from MCollective to Puppet orchestrator** on page 750
Puppet Enterprise 2018.1 is the last release to support MCollective. To prepare for these changes, migrate your MCollective work to Puppet orchestrator to automate tasks and create consistent, repeatable administrative processes.

**Removing MCollective** on page 752
Remove MCollective and its related files from the nodes in your infrastructure.

---

### MCollective
With MCollective in Puppet Enterprise, you can invoke many kinds of actions in parallel across any number of nodes. Several useful actions are available by default, and you can easily add and use new actions.

### Setting up MCollective
Because MCollective does not install with PE you must enable it.

**CAUTION:**
2018.1 is the last release to support Marionette Collective, also known as MCollective. While 2018.1 remains supported, will continue to address security issues for MCollective. Feature development has been discontinued. Future releases of will not include MCollective. For more information, see the Puppet Enterprise support lifecycle.

To prepare for these changes, migrate your MCollective work to orchestrator to automate tasks and create consistent, repeatable administrative processes. Use orchestrator to automate your workflows and take advantage of its integration with console and commands, APIs, role-based access control, and event tracking.

**Procedure**
- Before you install PE 2018.1 on the master, add the following parameter to your **pe.conf** file:

  "pe_install::disable_mco": false

**Related information**
- [Configuration parameters and the pe.conf file](#)
- A **pe.conf** file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.
- [Move from MCollective to Puppet orchestrator](#)
Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

**Actions and plugins**

MCollective tasks aren't quite like SSH, PowerShell, or other tools meant for running arbitrary shell code in an ad-hoc way.

MCollective is built around the idea of predefined **actions**—it is essentially a highly parallel **remote procedure call (RPC)** system.

**Actions** are distributed in **agent plugins**, which are bundles of several related actions.

- Many plugins are available by default; see Built-in actions.
- You can download or write new plugins and adding them to the engine.

**Invoking actions and filtering nodes**

The core concept of MCollective engine is **invoking actions**, in parallel, on a select group of nodes.

Typically you choose some nodes to operate on (usually with a **filter** that describes the desired fact values or classes), and specify an **action** and its **arguments**. MCollective then runs that action on the chosen nodes, and displays any data collected during the run.

You can invoke MCollective actions from the command line

In addition to the main action invocation interfaces, PE provides special interfaces for one of the most useful orchestration tasks, remotely controlling the agent and triggering runs.

**MCollective orchestration internals**

The following explain the internals of MCollective orchestration.

**Components**

MCollective consists of the following parts:

- The **pe-activemq** service (which runs on the Puppet master server) routes all messages.
- The **pe-mcollective** service (which runs on every agent node) listens for authorized commands and invokes actions in response. It relies on the available agent plugins for its set of possible actions.
- The **mco** command (available to the **peadmin** user account on the Puppet master server) can issue authorized commands to any number of nodes.

**Security**

The MCollective engine uses the same security model as the recommended "standard MCollective deployment." See the security model section for a more detailed rundown of these security measures.

In short, all commands and replies are encrypted in transit, and only a few authorized clients are permitted to send commands. By default, PE allows commands to be sent by:

- Read/write and admin users of the console
- Users able to log in to the Puppet master server with full administrator **sudo** privileges

If you extend MCollective by integrating external applications, you can limit the actions each application has access to by distributing policy files; see the configuring orchestration page for more details.

You can also allow additional users to log in as the **peadmin** user on the Puppet master, usually by distributing standard SSH public keys.
Network traffic

Every node (including all agent nodes, the Puppet master server, and the console) needs the ability to initiate connections to the Puppet master server over TCP port 61613. See the notes on firewall configuration for more details about PE's network traffic.

Invoking actions

You can invoke MCollective actions on both *nix and Windows machines, based on commands you run from the command line on a Linux-based Puppet master servers.

MCollective has its own section of the documentation site, which includes more complete details and examples for command line orchestration usage.

The following topics cover basic CLI usage and all PE-specific information; for more details, see the following pages from the full docs:

- MCollective command line usage
- Filtering

Logging into MCollective

To run commands, you must log in to the Puppet master server as the special peadmin user account, which PE creates during installation.

Note: PE does not support adding more MCollective user accounts.

By default, the peadmin account cannot log in with a password. We recommend you log in using sudo or by adding SSH keys to the peadmin account.

Logging in with sudo

Anyone able to log into the Puppet master server as an admin user with full root sudo privileges can become the peadmin user by running:

```bash
$ sudo -i -u peadmin
```

This is the default way to log in as the peadmin user. It means that MCollective commands can only be issued by the group of users who can fully control the Puppet master.

Logging in with SSH keys

If you wish to allow other users to run commands without giving them full control over the Puppet master, you can add their public SSH keys to peadmin's authorized keys file.

You can use Puppet's sshAuthorizedKey resource type to do this, or add keys manually to the /var/lib/peadmin/.ssh/authorized_keys file.

The mco command

All MCollective actions are invoked with the mco executable. The mco command always requires a subcommand to invoke actions.

Important: For security, the mco command relies on a config file (/var/lib/peadmin/.mcollective) which is only readable by the peadmin user. PE automatically configures and manages this file, so do not modify it.

mco Subcommands

The mco command has several subcommands, and it's possible to add more --- run mco help for a list of all available subcommands. The following default subcommands are available:
• **main**

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpc</td>
<td>This is the general purpose client, which can invoke actions from any agent plugin.</td>
</tr>
</tbody>
</table>

• **special purpose**

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet</td>
<td>These subcommands only invoke certain kinds of actions, but have some extra UI enhancements to make them easier to use than the equivalent <code>mco rpc</code> command.</td>
</tr>
<tr>
<td>package</td>
<td></td>
</tr>
<tr>
<td>service</td>
<td></td>
</tr>
</tbody>
</table>

• **Help and support subcommands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Displays help for subcommands.</td>
</tr>
<tr>
<td>plugin</td>
<td>The <code>mco plugin doc</code> command can display help for agent plugins.</td>
</tr>
<tr>
<td>completion</td>
<td>A helper for shell completion systems.</td>
</tr>
</tbody>
</table>

• **Inventory and reporting subcommands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping</td>
<td>Pings all matching nodes and reports on response times</td>
</tr>
<tr>
<td>facts</td>
<td>Displays a summary of values for a single fact across all systems</td>
</tr>
<tr>
<td>inventory</td>
<td>General reporting tool for nodes, collectives and subcollectives</td>
</tr>
<tr>
<td>find</td>
<td>Like ping, but doesn't report response times</td>
</tr>
</tbody>
</table>

**Getting help on the command line**

You can get information about subcommands, actions, and other plugins on the command line.

**Subcommand Help**

Use one of the following commands to get help for a specific subcommand:

```
$ mco help <SUBCOMMAND>
$ mco <SUBCOMMAND> --help
```

**List of plugins**

To get a list of the available plugins, which includes agent plugins, data query plugins, discovery methods, and validator plugins, run `mco plugin doc`.

**Agent plugin help**

Related actions are bundled together in agent plugins. For example, Puppet-related actions are all in the `puppet` plugin.
To get detailed info on a given plugin's actions and their required inputs, run:

```
$ mco plugin doc <PLUGIN>
```

If there is also a data plugin with the same name, you may need to prepend `agent/` to the plugin name to disambiguate:

```
$ mco plugin doc agent/<PLUGIN>
```

## Invoking actions

MCollective actions are invoked with either the general purpose `rpc` subcommand or one of the special-purpose subcommands. Note that unless you specify a filter, commands will be run on every server in your Puppet Enterprise deployment; make sure you know what will happen before confirming any potentially disruptive commands.

### The `rpc` subcommand

The most useful subcommand is `mco rpc`. This is the general purpose client, which can invoke actions from any agent plugin. See the list of built-in actions for more information about agent plugins.

**Example:**

```
$ mco rpc service restart service=httpd
```

The general form of an `mco rpc` command is:

```
$ mco rpc <AGENT PLUGIN> <ACTION> <INPUT>=<VALUE>
```

For a list of available agent plugins, actions, and their required inputs, see the list of built-in actions actions or the information on getting help.

### Special-purpose subcommands

Although `mco rpc` can invoke any action, sometimes a special-purpose application can provide a more convenient interface.

**Example:**

```
$ mco puppet runall 5
```

The `puppet` subcommand's special `runall` action is able to run many nodes without exceeding a certain load of concurrent runs. It does this by repeatedly invoking the `puppet agent's status` action, and only sending a `runonce` action to the next node if there's enough room in the concurrency limit.

This uses the same actions that the `mco rpc` command can invoke, but since `rpc` doesn't know that the output of the `status` action is relevant to the timing of the `runonce` action, it can't provide that improved UI.

Each special-purpose subcommand (`puppet`, `service`, and `package`) has its own CLI syntax. For example, `mco service` puts the name of the service before the action, to mimic the format of the more common platform-specific service commands:

```
$ mco service httpd status
```

Run `mco help <SUBCOMMAND>` to get specific help for each subcommand.

**Related information**

[Getting help on the command line](#) on page 728
You can get information about subcommands, actions, and other plugins on the command line.

List of built-in actions on page 735

MCollective comes with several built-in actions. You invoke these actions from the command line.

Filtering actions

By default, orchestration actions affect all managed nodes. You can limit any action to a smaller set of nodes by specifying a filter.

For example, the following filter checks the status of the pe-nginx service running on the node defined as the console.

$ mco service pe-nginx status --with-fact fact_is_puppetconsole=true

Note: For more details about filters, see the following pages from the MCollective docs:

• MCollective CLI Usage: Filters
• Filtering

All command line actions can accept the same filter options, which are listed under the "Host Filters" section of any mco help <SUBCOMMAND> text:

Host Filters
- W, --with FILTER                   Combined classes and facts filter
- S, --select FILTER                 Compound filter combining facts and classes
- F, --wf, --with-fact fact=val      Match hosts with a certain fact
- C, --wc, --with-class CLASS        Match hosts with a certain config management class
- A, --wa, --with-agent AGENT        Match hosts with a certain agent
- I, --wi, --with-identity IDENT     Match hosts with a certain configured identity

Each type of filter lets you specify a type of metadata and a desired value. The action will only run on nodes where that data has that desired value.

Any number of fact, class, and agent filters can also be combined in a single command; this will make it so nodes must match every filter to run the action.

Matching strings and regular expressions

Filter values are usually simple strings. These must match exactly and are case-sensitive.

Most filters can also accept regular expressions as their values; these are surrounded by forward slashes, and are interpreted as standard Ruby regular expressions. (You can even turn on various options for a subpattern, such as case insensitivity --- -F "osfamily=/(?i:redhat)/".) Unlike plain strings, they accept partial matches.

Filtering by identity

A node’s "identity" is the same as its certname, as specified during installation. Identities will almost always be unique per node.

$ mco puppet runonce -I web3balancer.example.com

• You can use the -I or --with-identity option multiple times to create a filter that matches multiple specific nodes.
• You cannot combine the identity filter with other filter types.
• The identity filter accepts regular expressions.
Filtering by fact, class, and agent

• **Facts** are the standard Puppet facts, which are available in your Puppet manifests. A list of the core facts is available here. Use the `-F` or `--with-fact` option with a `fact=value` pair to filter on facts.

• **Classes** are the Puppet classes that are assigned to a node. This includes classes assigned in the console, assigned via Hiera, declared in `site.pp`, or declared indirectly by another class. Use the `--class` or `--with-class` option with a class name to filter on classes.

• **Agents** are MCollective agent plugins. PE's default plugins are available on every node, so filtering by agent makes more sense if you are distributing custom plugins to only a subset of your nodes. For example, if you made an emergency change to a custom plugin that you distribute with Puppet, you could filter by agent to trigger an immediate run on all affected systems. (mco puppet runall 5 -A my_agent) Use the `-A` or `--with-agent` option to filter on agents.

Since mixing classes and facts is so common, you can also use the `-w` or `--with` option to supply a mixture of class names and `fact=value` pairs.

Compound "select" filters

The `-S` or `--select` option accepts arbitrarily complex filters. Like `-w`, it can accept a mixture of class names and `fact=value` pairs, but it has two extra tricks:

• **Boolean logic**

  The `-w` filter always combines facts and classes with "and" logic --- nodes must match all of the criteria to match the filter.

  The `-S` filter lets you combine values with nested Boolean "and"/"or"/"not" logic:

  ```
  $ mco service httpd restart -S "((customer=acme and osfamily=RedHat) or domain=acme.com) and /apache/"
  ```

• **Data plugins**

  In addition, the `-S` filter lets you use data plugin queries as an additional kind of metadata.

  Data plugins can be tricky, but are very powerful. To use them effectively, you must:

  1. Check the list of data plugins with mco plugin doc.
  2. Read the help for the data plugin you want to use, with mco plugin doc data/<NAME>. Note any required input and the available outputs.
  3. Use the rpcutil plugin's get_data action on a single node to check the format of the output you're interested in. This action requires source (the plugin name) and query (the input) arguments:

     ```
     $ mco rpc rpcutil get_data source="fstat" query="/etc/hosts" -I web01
     ```

     This will show all of the outputs for that plugin and input on that node.

  4. Construct a query fragment of the format `<PLUGIN>("<INPUT>").<OUTPUT>=<VALUE>` --- note the parentheses, the fact that the input must be in quotes, the .output notation, and the equals sign. Make sure the value you're searching for matches the expected format, which you saw when you did your test query.

  5. Use that fragment as part of a `-S` filter:

     ```
     $ mco find -S "fstat('/etc/hosts').md5=/baa3772104/ and osfamily=RedHat"
     ```

You can specify multiple data plugin query fragments per `-S` filter.

The MCollective documentation includes a page on writing custom data plugins. Installing custom data plugins is similar to installing custom agent plugins; see Adding new actions for details.

© 2020 Puppet, Inc.
Testing filters with `mco find`

Before invoking any potentially disruptive action, like a service restart, you should test the filter with `mco find` or `mco ping`, to make sure your command will act on the nodes you expect.

Batching and limiting actions

By default, actions run simultaneously on all of the targeted nodes.

This is fast and powerful, but is sometimes not what you want:

- Sometimes you want the option to cancel out of an action with control-C before all nodes have run it.
- Sometimes, like when retrieving inventory data, you want to run a command on just a sample of nodes and don't need to see the results from everything that matches the filter.
- Certain actions may consume limited capacity on a shared resource (such as the Puppet master server), and invoking them on a "thundering herd" of nodes can disrupt that resource.

In these cases, you can **batch** actions, to run all of the matching nodes in a controlled series, or **limit** them, to run only a subset of the matching nodes.

**Batching**

- Use the `--batch <SIZE>` option to invoke an action on only `<SIZE>` nodes at once. PE will invoke it on the first `<SIZE>` nodes, wait briefly, invoke it on the next batch, and so on.
- Use the `--batch-sleep <SECONDS>` option to control how long PE should sleep between batches.

**Limiting**

- Use the `--limit <COUNT>` option to invoke an action on only `<COUNT>` matching nodes. `<COUNT>` can be an absolute number or a percentage. The nodes will be chosen randomly.
- Use the `-1` or `--one` option to invoke an action on just one matching node, chosen randomly.

Controlling Puppet

PE's configuration management features rely on the Puppet agent service, which runs on every node and fetches configurations from the Puppet master server.

By default, the agent idles in the background and performs a run every 30 minutes, but MCollective can give complete control over this behavior.

**Note:** The MCollective engine cannot trigger a node's *very first* agent run. A node's first run will happen automatically within 30 minutes after you sign its certificate.

**Related information**

Logging into MCollective on page 727

To run commands, you must log in to the Puppet master server as the special `peadmin` user account, which PE creates during installation.

Running Puppet on demand

Use the `runonce` action to trigger an immediate Puppet run on a few nodes. If you need to run Puppet on many nodes (more than ten), you should see the "many nodes" section below.

**Behavior differences: Running vs. stopped**

You can trigger on-demand runs whether the `puppet` service is running or stopped, but on *nix nodes these cases will behave slightly differently:

- When the service is `running`, all of the selected nodes will begin a run *immediately*, and you cannot specify any special options like `noop` or `tags`; they will be ignored. This behavior is usually fine but sometimes undesirable.
When the service is **stopped**, the selected nodes will randomly stagger the start of their runs ("splay") over a default interval of **two minutes**. If you wish, you can specify special options, including a longer interval ("splaylimit"). You can also set the **force** option to **true** if you want the selected nodes to start immediately. This behavior is more flexible and resilient.

This difference only affects *nix nodes; Windows nodes always behave like a **stopped** *nix node.

**Triggering on-demand runs on the command line**

While logged in to the Puppet master server as `peadmin`, run the `mco puppet runonce` command.

```bash
$ mco puppet runonce -I web01.example.com -I web02.example.com
$ mco puppet runonce -F kernelversion=2.6.32
```

Be sure to specify a filter to limit the number of nodes; you should generally invoke this action on fewer than 10 nodes at a time, especially if the agent service is running and you cannot specify extra options (see above).

**Additional command options**

If the agent service is stopped (on affected *nix nodes; see above), you can change the way Puppet runs with command line options. You can see a list of these by running `mco puppet --help`.

- `--force` Bypass splay options when running
- `--server SERVER` Connect to a specific server or port
- `--tags, --tag TAG` Restrict the run to specific tags
- `--noop` Do a no-op run
- `--no-noop` Do a run with no-op disabled
- `--environment ENVIRONMENT` Place the node in a specific environment for this run
- `--splay` Splay the run by up to splaylimit seconds
- `--no-splay` Do a run with splay disabled
- `--splaylimit SECONDS` Maximum splay time for this run if splay is set
- `--ignoreschedules` Disable schedule processing

**Running Puppet on many nodes in a controlled series**

If you want to trigger a run on a large number of nodes—more than ten—the `runonce` action isn't always the best tool. You can splay or batch the runs, but this requires you to guess how long each run is going to take, and a wrong guess can either waste time or temporarily overwhelm the Puppet master server.

Instead, use the special `runall` action of the `mco puppet` subcommand.

```bash
$ mco puppet runall 5 -F operatingsystem=CentOS -F operatingsystemrelease=6.4
```

This action requires an argument, which must be the number of nodes allowed to run at once. It invokes a run on that many nodes, then only starts the next node when one has finished. This prevents your Puppet master from being overwhelmed by the herd and will delay only as long as is necessary. The ideal concurrency will vary from site to site, depending on how powerful your Puppet master server is and how complex your configurations are.

The `runall` action can take extra options like `--noop` as described for the `runonce` action; however, note that restrictions still apply for *nix nodes where the pe-puppet service is running.

**Enabling and disabling Puppet agent**

Disabling Puppet will block all runs, including both scheduled and on-demand runs. This is usually used while you investigate some kind of problem. Use the `enable` and `disable` actions of the `puppet` plugin.

The `disable` action accepts an optional reason for the lockdown; take advantage of this to keep your colleagues informed. The reason will be shown when checking Puppet's status on those nodes.
After a node has been disabled for an hour, it will appear as "unresponsive" in the console's node views, and will stay that way until it is re-enabled.

**Enabling and disabling Puppet agent on the command line**

While logged in to the Puppet master server as `peadmin`, run `mco puppet disable` or `mco puppet enable` with or without a filter.

**Example:** You noticed runs failing on a load balancer and expect they'll start failing on the other ones too:

```
$ mco puppet disable "Investigating a problem with the haproxy module. -NF" -C /haproxy/
```

**Starting and stopping the Puppet agent service**

You can start or stop the `puppet` service with the `start` and `stop` actions of the `service` plugin. This can be useful if you need to do no-op runs, or if you wish to stop all scheduled runs and only run agents on demand.

**Starting and stopping the agent service on the command line**

While logged in to the Puppet master server as `peadmin`, run `mco service puppet stop` or `mco service puppet start` with or without a filter.

**Example:** To prepare all web servers for a manifest update and no-op, run:

```
$ mco service puppet stop -C /apache/
```

**Viewing the Puppet agent's status**

The agent can be in various states. The MCollective engine lets you check the current status on any number of nodes.

**Viewing the agent's state on the command line**

While logged in to the Puppet master server as `peadmin`, run `mco puppet status` with or without a filter. This returns an abbreviated status for each node and a summarized breakdown of how many nodes are in which conditions.

```
$ mco puppet status
```

**Viewing disable messages**

The one thing `mco puppet status` doesn't show is the reason why the agent was disabled. If you're checking up on disabled nodes, you can get a more raw view of the status by running `mco rpc puppet status` instead. This will display the reason in the `Lock Message` field.

**Example:** Get the detailed status for every disabled node, using the `puppet` data plugin:

```
$ mco rpc puppet status -S "puppet().enabled=false"
```

**Viewing statistics about recent runs**

Puppet keeps records of the last run, including the amount of time spent per resource type, the number of changes, number of simulated changes, and the time since last run. You can retrieve and summarize these statistics with the MCollective engine.

**Viewing statistics on the command line**

You can view population summary graphs or detailed statistics.
• Population summary graphs

You can get sparkline graphs for the last run statistics across all your nodes with the `mco puppet summary` command. This shows the distribution of your nodes, so you can see whether a significant group is taking notably longer or seeing more changes.

```shell
$ mco puppet summary
Summary statistics for 10 nodes:

      Total resources: ################################ min: 93.0  max: 155.0
      Out Of Sync resources: ################################ min: 0.0  max: 0.0
      Failed resources: ################################     min: 0.0  max: 0.0
      Changed resources: ################################     min: 0.0  max: 0.0
  Config Retrieval time (seconds): ################################ min: 1.9  max: 5.8
       Total run-time (seconds): ################################ min: 2.2  max: 6.7
Time since last run (seconds): ################################ min: 314.0 max: 23.4k
```

• Detailed statistics

While logged in to the Puppet master server as `peadmin`, run `mco rpc puppet last_run_summary` with or without a filter. This returns detailed run statistics for each node. (Note that this uses the `rpc` subcommand instead of the `puppet` subcommand.)

### List of built-in actions

MCollective in comes with several built-in actions. You invoke these actions from the command line.

#### MCollective actions and plugins

Sets of related actions are bundled together as MCollective agent plugins. Every action is part of a plugin.

A default PE install includes the package, puppet, puppetral, rpcutil, and service plugins. See the table of contents above for an outline of each plugin's actions; click an action for details about its inputs, effects, and outputs.

You can easily add new actions by distributing custom agent plugins to your nodes. See Adding actions for details.

#### The package plugin

Install and uninstall software packages.

Actions: `apt_checkupdates, apt_update, checkupdates, install, purge, status, uninstall, update, yum_checkupdates, yum_clean`

- **apt_checkupdates**: Check for APT updates

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>(Appears as &quot;Exit Code&quot; on CLI). The exitcode from the apt command</td>
</tr>
<tr>
<td>outdated_packages</td>
<td>(Appears as &quot;Outdated Packages&quot; on CLI) Outdated packages</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from APT</td>
</tr>
</tbody>
</table>
• **apt_update**: Update the apt cache

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>(Appears as &quot;Exit Code&quot; on CLI) The exitcode from the apt-get command</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from apt-get</td>
</tr>
</tbody>
</table>

• **checkupdates**: Check for updates

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>(Appears as &quot;Exit Code&quot; on CLI) The exitcode from the package manager command</td>
</tr>
<tr>
<td>outdated_packages</td>
<td>(Appears as &quot;Outdated Packages&quot; on CLI) Outdated packages</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from Package Manager</td>
</tr>
<tr>
<td>package_manager</td>
<td>(Appears as &quot;Package Manager&quot; on CLI) The detected package manager</td>
</tr>
</tbody>
</table>

• **install**: Install a package

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>package <em>(required)</em></td>
<td>Package to install Type: string, Format/Validation: shellsafe, Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>(Appears as &quot;Arch&quot; on CLI) Package architecture</td>
</tr>
<tr>
<td>ensure</td>
<td>(Appears as &quot;Ensure&quot; on CLI) Full package version</td>
</tr>
<tr>
<td>epoch</td>
<td>(Appears as &quot;Epoch&quot; on CLI) Package epoch number</td>
</tr>
<tr>
<td>name</td>
<td>(Appears as &quot;Name&quot; on CLI) Package name</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from the package manager</td>
</tr>
<tr>
<td>provider</td>
<td>(Appears as &quot;Provider&quot; on CLI) Provider used to retrieve information</td>
</tr>
<tr>
<td>release</td>
<td>(Appears as &quot;Release&quot; on CLI) Package release number</td>
</tr>
<tr>
<td>version</td>
<td>(Appears as &quot;Version&quot; on CLI) Version number</td>
</tr>
</tbody>
</table>

• **purge**: Purge a package

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>package <em>(required)</em></td>
<td>Package to purge Type: string, Format/Validation: shellsafe, Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>(Appears as &quot;Arch&quot; on CLI) Package architecture</td>
</tr>
<tr>
<td>ensure</td>
<td>(Appears as &quot;Ensure&quot; on CLI) Full package version</td>
</tr>
<tr>
<td>epoch</td>
<td>(Appears as &quot;Epoch&quot; on CLI) Package epoch number</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>provider</td>
</tr>
<tr>
<td>release</td>
</tr>
<tr>
<td>version</td>
</tr>
</tbody>
</table>

- **status**: Get the status of a package

### Inputs

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>package (required)</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
</tr>
<tr>
<td>ensure</td>
</tr>
<tr>
<td>epoch</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>provider</td>
</tr>
<tr>
<td>release</td>
</tr>
<tr>
<td>version</td>
</tr>
</tbody>
</table>

- **uninstall**: Uninstall a package

### Inputs

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>package (required)</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
</tr>
<tr>
<td>ensure</td>
</tr>
<tr>
<td>epoch</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>provider</td>
</tr>
<tr>
<td>release</td>
</tr>
<tr>
<td>version</td>
</tr>
</tbody>
</table>
• update: Update a package

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>package</strong> (required)</td>
<td>Package to update <strong>Type:</strong> string <strong>Format/Validation:</strong> shellsafe <strong>Length:</strong> 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>(Appears as &quot;Arch&quot; on CLI) Package architecture</td>
</tr>
<tr>
<td>ensure</td>
<td>(Appears as &quot;Ensure&quot; on CLI) Full package version</td>
</tr>
<tr>
<td>epoch</td>
<td>(Appears as &quot;Epoch&quot; on CLI) Package epoch number</td>
</tr>
<tr>
<td>name</td>
<td>(Appears as &quot;Name&quot; on CLI) Package name</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from the package manager</td>
</tr>
<tr>
<td>provider</td>
<td>(Appears as &quot;Provider&quot; on CLI) Provider used to retrieve information</td>
</tr>
<tr>
<td>release</td>
<td>(Appears as &quot;Release&quot; on CLI) Package release number</td>
</tr>
<tr>
<td>version</td>
<td>(Appears as &quot;Version&quot; on CLI) Version number</td>
</tr>
</tbody>
</table>

• yum_checkupdates: Check for YUM updates

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>(Appears as &quot;Exit Code&quot; on CLI) The exitcode from the yum command</td>
</tr>
<tr>
<td>outdated_packages</td>
<td>(Appears as &quot;Outdated Packages&quot; on CLI) Outdated packages</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from Yum</td>
</tr>
</tbody>
</table>

• yum_clean: Clean the YUM cache

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>One of the various supported clean modes <strong>Type:</strong> list <strong>Valid Values:</strong> all, headers, packages, metadata, dbcache, plugins, expire-cache</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitcode</td>
<td>(Appears as &quot;Exit Code&quot; on CLI) The exitcode from the yum command</td>
</tr>
<tr>
<td>output</td>
<td>(Appears as &quot;Output&quot; on CLI) Output from Yum</td>
</tr>
</tbody>
</table>

**The puppet plugin**

Run Puppet agent, get its status, and enable/disable it

Actions: disable, enable, last_run_summary, resource, runonce, status
• disable: Disable the Puppet agent

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Supply a reason for disabling the Puppet agent <strong>Type:</strong> string <strong>Format/Validation:</strong> <code>shellsafe</code> <strong>Length:</strong> 120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>(Appears as &quot;Enabled&quot; on CLI) Is the agent currently locked</td>
</tr>
<tr>
<td>status</td>
<td>(Appears as &quot;Status&quot; on CLI) Status</td>
</tr>
</tbody>
</table>

• enable: Enable the Puppet agent

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>(Appears as &quot;Enabled&quot; on CLI) Is the agent currently locked</td>
</tr>
<tr>
<td>status</td>
<td>(Appears as &quot;Status&quot; on CLI) Status</td>
</tr>
</tbody>
</table>

• last_run_summary: Get the summary of the last Puppet run

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>changed_resources</td>
<td>(Appears as &quot;Changed Resources&quot; on CLI) Resources that were changed</td>
</tr>
<tr>
<td>config_retrieval_time</td>
<td>(Appears as &quot;Config Retrieval Time&quot; on CLI) Time taken to retrieve the catalog from the master</td>
</tr>
<tr>
<td>config_version</td>
<td>(Appears as &quot;Config Version&quot; on CLI) Puppet config version for the previously applied catalog</td>
</tr>
<tr>
<td>failed_resources</td>
<td>(Appears as &quot;Failed Resources&quot; on CLI) Resources that failed to apply</td>
</tr>
<tr>
<td>lastrun</td>
<td>(Appears as &quot;Last Run&quot; on CLI) When the Agent last applied a catalog in local time</td>
</tr>
<tr>
<td>out_of_sync_resources</td>
<td>(Appears as &quot;Out of Sync Resources&quot; on CLI) Resources that were not in desired state</td>
</tr>
<tr>
<td>since_lastrun</td>
<td>(Appears as &quot;Since Last Run&quot; on CLI) How long ago did the Agent last apply a catalog in local time</td>
</tr>
<tr>
<td>summary</td>
<td>(Appears as &quot;Summary&quot; on CLI) Summary data as provided by Puppet</td>
</tr>
<tr>
<td>total_resources</td>
<td>(Appears as &quot;Total Resources&quot; on CLI) Total resources managed on a node</td>
</tr>
<tr>
<td>total_time</td>
<td>(Appears as &quot;Total Time&quot; on CLI) Total time taken to retrieve and process the catalog</td>
</tr>
<tr>
<td>type_distribution</td>
<td>(Appears as &quot;Type Distribution&quot; on CLI) Resource counts per type managed by Puppet</td>
</tr>
</tbody>
</table>

• resource: Evaluate Puppet RAL resources

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name (required)</td>
<td>Resource Name <strong>Type:</strong> string <strong>Format/Validation:</strong> <code>^\s+.+$</code> <strong>Length:</strong> 150</td>
</tr>
</tbody>
</table>
### Inputs

<table>
<thead>
<tr>
<th>Definition</th>
<th>Type: string <strong>Format/Validation:</strong> ^+.+$ <strong>Length:</strong> 50</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong> <em>(required)</em></td>
<td>Resource Type</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Definition</th>
<th><em>(Appears as &quot;Changed&quot; on CLI) Was a change applied based on the resource</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>changed</strong></td>
<td><em>(Appears as &quot;Result&quot; on CLI) The result from the Puppet resource</em></td>
</tr>
</tbody>
</table>

- **runonce:** Invoke a single Puppet run

### Inputs

<table>
<thead>
<tr>
<th>Definition</th>
<th><em>(Which Puppet environment to run)</em> <strong>Type:</strong> string <strong>Format/Validation:</strong> puppet_variable <strong>Length:</strong> 50</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>environment</strong></td>
<td><em>(Will force a run immediately else is subject to default splay time)</em> <strong>Type:</strong> boolean</td>
</tr>
<tr>
<td><strong>force</strong></td>
<td><em>(Do a Puppet dry run)</em> <strong>Type:</strong> boolean</td>
</tr>
<tr>
<td><strong>noop</strong></td>
<td><em>(Address and port of the Puppet Master in server:port format)</em> <strong>Type:</strong> string <strong>Format/Validation:</strong> puppet_server_address <strong>Length:</strong> 50</td>
</tr>
<tr>
<td><strong>server</strong></td>
<td><em>(Sleep for a period before initiating the run)</em> <strong>Type:</strong> boolean</td>
</tr>
<tr>
<td><strong>splay</strong></td>
<td><em>(Maximum amount of time to sleep before run)</em> <strong>Type:</strong> number</td>
</tr>
<tr>
<td><strong>splaylimit</strong></td>
<td><em>(Restrict the Puppet run to a comma list of tags)</em> <strong>Type:</strong> strings <strong>Format/Validation:</strong> puppet_tags <strong>Length:</strong> 120</td>
</tr>
<tr>
<td><strong>tags</strong></td>
<td><em>(Appears as &quot;Summary&quot; on CLI) Summary of command run</em></td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Definition</th>
<th><em>(Appears as &quot;Summary&quot; on CLI) Summary of command run</em></th>
</tr>
</thead>
</table>

- **status:** Get the current status of the Puppet agent

### Outputs

<table>
<thead>
<tr>
<th>Definition</th>
<th><em>(Appears as &quot;Applying&quot; on CLI) Is a catalog being applied</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>applying</strong></td>
<td><em>(Appears as &quot;Daemon Running&quot; on CLI) Is the Puppet agent daemon running on this system</em></td>
</tr>
<tr>
<td><strong>daemon_present</strong></td>
<td><em>(Appears as &quot;Lock Message&quot; on CLI) Message supplied when agent was disabled</em></td>
</tr>
<tr>
<td><strong>disable_message</strong></td>
<td><em>(Appears as &quot;Enabled&quot; on CLI) Is the agent currently locked</em></td>
</tr>
<tr>
<td><strong>enabled</strong></td>
<td><em>(Appears as &quot;Idling&quot; on CLI) Is the Puppet agent daemon running but not doing any work</em></td>
</tr>
<tr>
<td><strong>idling</strong></td>
<td><em>(Appears as &quot;Last Run&quot; on CLI) When the Agent last applied a catalog in local time</em></td>
</tr>
<tr>
<td><strong>lastrun</strong></td>
<td><em>(Appears as &quot;Status&quot; on CLI)</em></td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
The **puppetral plugin**

View resources with Puppet's resource abstraction layer

Actions: find, search

- **find**: Get the attributes and status of a resource

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>title</strong> <em>(required)</em></td>
<td>Name of resource to check <em>Type: string Format/Validation: . Length: 90</em></td>
</tr>
<tr>
<td><strong>type</strong> <em>(required)</em></td>
<td>Type of resource to check <em>Type: string Format/Validation: . Length: 90</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>exported</strong></td>
<td>(Appears as &quot;Exported&quot; on CLI) Boolean flag indicating export status</td>
</tr>
<tr>
<td><strong>managed</strong></td>
<td>(Appears as &quot;Managed&quot; on CLI) Flag indicating managed status</td>
</tr>
<tr>
<td><strong>parameters</strong></td>
<td>(Appears as &quot;Parameters&quot; on CLI) Parameters of the inspected resource</td>
</tr>
<tr>
<td><strong>tags</strong></td>
<td>(Appears as &quot;Tags&quot; on CLI) Tags of the inspected resource</td>
</tr>
<tr>
<td><strong>title</strong></td>
<td>(Appears as &quot;Title&quot; on CLI) Title of the inspected resource</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>(Appears as &quot;Type&quot; on CLI) Type of the inspected resource</td>
</tr>
</tbody>
</table>

- **search**: Get detailed info for all resources of a given type

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong> <em>(required)</em></td>
<td>Type of resource to check <em>Type: string Format/Validation: . Length: 90</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>result</strong></td>
<td>(Appears as &quot;Result&quot; on CLI) The values of the inspected resources</td>
</tr>
</tbody>
</table>

The **rpcutil plugin**

General helpful actions that expose stats and internals to SimpleRPC clients

Actions: agent_inventory, collective_info, daemon_stats, get_config_item, get_data, get_fact, inventory, ping

© 2020 Puppet, Inc.
• **agent_inventory:** Inventory of all agents on the server

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>agents</td>
<td>(Appears as &quot;Agents&quot; on CLI) List of agents on the server</td>
</tr>
</tbody>
</table>

• **collective_info:** Info about the main and sub collectives

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectives</td>
<td>(Appears as &quot;All Collectives&quot; on CLI) All Collectives</td>
</tr>
<tr>
<td>main_collective</td>
<td>(Appears as &quot;Main Collective&quot; on CLI) The main Collective</td>
</tr>
</tbody>
</table>

• **daemon_stats:** Get statistics from the running daemon (no inputs)

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>agents</td>
<td>(Appears as &quot;Agents&quot; on CLI) List of agents loaded</td>
</tr>
<tr>
<td>configfile</td>
<td>(Appears as &quot;Config File&quot; on CLI) Config file used to start the daemon</td>
</tr>
<tr>
<td>filtered</td>
<td>(Appears as &quot;Failed Filter&quot; on CLI) Didn't pass filter checks</td>
</tr>
<tr>
<td>passed</td>
<td>(Appears as &quot;Passed Filter&quot; on CLI) Passed filter checks</td>
</tr>
<tr>
<td>pid</td>
<td>(Appears as &quot;PID&quot; on CLI) Process ID of the daemon</td>
</tr>
<tr>
<td>replies</td>
<td>(Appears as &quot;Replies&quot; on CLI) Replies sent back to clients</td>
</tr>
<tr>
<td>starttime</td>
<td>(Appears as &quot;Start Time&quot; on CLI) Time the server started</td>
</tr>
<tr>
<td>threads</td>
<td>(Appears as &quot;Threads&quot; on CLI) List of threads active in the daemon</td>
</tr>
<tr>
<td>times</td>
<td>(Appears as &quot;Times&quot; on CLI) Processor time consumed by the daemon</td>
</tr>
<tr>
<td>total</td>
<td>(Appears as &quot;Total Messages&quot; on CLI) Total messages received</td>
</tr>
<tr>
<td>ttlexpired</td>
<td>(Appears as &quot;TTL Expired&quot; on CLI) Messages that did pass TTL checks</td>
</tr>
<tr>
<td>unvalidated</td>
<td>(Appears as &quot;Failed Security&quot; on CLI) Messages that failed security validation</td>
</tr>
<tr>
<td>validated</td>
<td>(Appears as &quot;Security Validated&quot; on CLI) Messages that passed security validation</td>
</tr>
<tr>
<td>version</td>
<td>(Appears as &quot;Version&quot; on CLI) MCollective Version</td>
</tr>
</tbody>
</table>
• **get_config_item**: Get the active value of a specific config property

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>item</strong> (required)</td>
<td>The item to retrieve from the server <strong>Type</strong>: string <strong>Format/Validation</strong>: ^.+$ <strong>Length</strong>: 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>item</strong></td>
<td>(Appears as &quot;Property&quot; on CLI) The config property being retrieved</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>(Appears as &quot;Value&quot; on CLI) The value that is in use</td>
</tr>
</tbody>
</table>

• **get_data**: Get data from a data plugin

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>query</strong></td>
<td>The query argument to supply to the data plugin <strong>Type</strong>: string <strong>Format/Validation</strong>: ^.+$ <strong>Length</strong>: 50</td>
</tr>
<tr>
<td><strong>source</strong> (required)</td>
<td>The data plugin to retrieve information from <strong>Type</strong>: string <strong>Format/Validation</strong>: ^\w+$ <strong>Length</strong>: 50</td>
</tr>
</tbody>
</table>

• **get_fact**: Retrieve a single fact from the fact store

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fact</strong> (required)</td>
<td>The fact to retrieve <strong>Type</strong>: string <strong>Format/Validation</strong>: ^[\w-.]+$ <strong>Length</strong>: 40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fact</strong></td>
<td>(Appears as &quot;Fact&quot; on CLI) The name of the fact being returned</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>(Appears as &quot;Value&quot; on CLI) The value of the fact</td>
</tr>
</tbody>
</table>

• **inventory**: System Inventory

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>agents</strong></td>
<td>(Appears as &quot;Agents&quot; on CLI) List of agent names</td>
</tr>
<tr>
<td><strong>classes</strong></td>
<td>(Appears as &quot;Classes&quot; on CLI) List of classes on the system</td>
</tr>
<tr>
<td><strong>collectives</strong></td>
<td>(Appears as &quot;All Collectives&quot; on CLI) All Collectives</td>
</tr>
<tr>
<td><strong>data_plugins</strong></td>
<td>(Appears as &quot;Data Plugins&quot; on CLI) List of data plugin names</td>
</tr>
<tr>
<td><strong>facts</strong></td>
<td>(Appears as &quot;Facts&quot; on CLI) List of facts and values</td>
</tr>
<tr>
<td><strong>main_collective</strong></td>
<td>(Appears as &quot;Main Collective&quot; on CLI) The main Collective</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td>(Appears as &quot;Version&quot; on CLI) MCollective Version</td>
</tr>
</tbody>
</table>

• **ping**: Responds to requests for PING with PONG

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pong</strong></td>
<td>(Appears as &quot;Timestamp&quot; on CLI) The local timestamp</td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
The service plugin
Start and stop system services

Actions: restart, start, status, stop

- **restart**: Restart a service

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service (required)</td>
<td>The service to restart Type: string Format/Validation:service_name Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>(Appears as &quot;Service Status&quot; on CLI) The status of the service after restarting</td>
</tr>
</tbody>
</table>

- **start**: Start a service

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service (required)</td>
<td>The service to start Type: string Format/Validation:service_name Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>(Appears as &quot;Service Status&quot; on CLI) The status of the service after starting</td>
</tr>
</tbody>
</table>

- **status**: Gets the status of a service

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service (required)</td>
<td>The service to get the status for Type: string Format/Validation:service_name Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>(Appears as &quot;Service Status&quot; on CLI) The status of the service after starting</td>
</tr>
</tbody>
</table>

- **stop**: Stop a service

<table>
<thead>
<tr>
<th>Input</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>service (required)</td>
<td>The service to stop Type: string Format/Validation:service_name Length: 90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>(Appears as &quot;Service Status&quot; on CLI) The status of the service after stopping</td>
</tr>
</tbody>
</table>
Adding actions and plugins to PE

You can extend PE's MCollective engine by adding new actions. Actions are distributed in agent plugins, which are bundles of several related actions. You can write your own agent plugins (or download ones created by other people), and use PE to install and configure them on your nodes.

Agent plugin components

Agent plugins consist of two parts: a .rb file containing the MCollective agent code and a .ddl file containing a description of plugin's actions, inputs, and outputs.

Every agent node that will be using this plugin needs both files. The Puppet master node and console node each need the .ddl file.

Note: Additionally, some agent plugins may be part of a bundle of related plugins, which may include new subcommands, data plugins, and more. A full list of plugin types and the nodes they should be installed on is available here. Note that "servers" refers to PE agent nodes and "clients" refers to the Puppet master and console nodes.

Agent plugin distribution

Not every agent node needs to use every plugin—MCollective is built to gracefully handle an inconsistent mix of plugins across nodes.

This means you can distribute special-purpose plugins to only the nodes that need them, without worrying about securing them on irrelevant nodes. Nodes that don't have a given plugin will ignore its actions, and you can also filter commands by the list of installed plugins.

Getting new plugins

You can write your own MCollective plugins, or download ones written by other people.

Downloading agent plugins

There isn't a central repository of agent plugins, but there are several good places to start looking:

- A list of the plugins released by Puppet is available here.
- If you use Nagios, the NRPE plugin is a good first plugin to install.
- Searching GitHub for “mcollective agent” will turn up many plugins, including ones for vmware_tools, libvirt, junk filters in iptables, and more.

Writing agent plugins

Most people who use MCollective heavily will want custom actions tailored to the needs of their own infrastructure. You can get these by writing new agent plugins in Ruby.

The MCollective documentation has instructions for writing agent plugins:

- Writing agent plugins
- Writing DDL files
- Aggregating replies for better command line interfaces

Additionally, you can learn a lot by reading the code of PE's built-in plugins. These are located in the /opt/puppet/libexec/mcollective/mcollective/ directory on any *nix node.

Overview of plugin installation process

Since actions need to be installed on many nodes, and since installing or upgrading an agent should always restart the mcollective service, you should use Puppet to install agent plugins.
Note: The full MCollective documentation includes a guide to installing plugins. Puppet Enterprise users must use the "copy into libdir" installation method. The remainder of this topic goes into more detail about using this method with Puppet Enterprise.

To install a new agent plugin, you must write a Puppet module that does the following things:

- On agent nodes: copy the plugin's .rb and .ddl files into the mcollective/agent subdirectory of MCollective's libdir. This directory's location varies between *nix and Windows nodes.
- On the console and Puppet master nodes: if you will not be installing this plugin on every agent node, copy the plugin's .ddl file into the mcollective/agent subdirectory of MCollective's libdir.
- If there are any other associated plugins included (such as data or validator plugins), copy them into the proper libdir subdirectories on agent nodes, the console node, and the Puppet master node.
- If any of these files change, restart the mcollective service, which is managed by the puppet_enterprise_mcollective module.

To accomplish these, you will need to write some limited interaction with the puppet_enterprise_mcollective module, which is part of PE's implementation.

Step 1: Create a module for your plugins

You have several options for laying this out:

- **One class for all of your custom plugins.** This works fine if you have a limited number of plugins and will be installing them on every agent node.
- **One module with several classes for individual plugins or groups of plugins.** This is good for installing certain plugins on only some of your agent nodes --- you can split specialized plugins into a pair of mcollective_plugins::<name>::agent and mcollective_plugins::<name>::client classes, and assign the former to the affected agent nodes and the latter to the console and Puppet master nodes.
- **A new module for each plugin.** This is maximally flexible, but can sometimes get cluttered.

After the module is created, **put the plugin files into its files/ directory.**

Step 2: Create relationships and set variables

For any class that will be installing plugins on agent nodes, you should put the following four lines near the top of the class definition:

```puppet
Class['puppet_enterprise::mcollective::server::plugins'] ->
Class[$title] ~> Service['mcollective']
  include puppet_enterprise::params
  $plugin_basedir = "${puppet_enterprise::params::mco_plugin_userdir}/mcollective"
  $mco_etc = $puppet_enterprise::params::mco_etc
```

This will do the following:

- Ensure that the necessary plugin directories already exist before we try to put files into them. (In certain cases, these directories are managed by variables in the puppet_enterprise::params class.)
- Restart the mcollective service whenever new plugins are installed or upgraded. (This service resource is declared in the puppet_enterprise::mcollective::server class.)
- Set variables that will correctly refer to the plugins directory and configuration directory on both *nix and Windows nodes.

Note: The Class[$title] notation seen above is a resource reference to the class that contains this statement; it uses the $title variable, which always contains the name of the surrounding container.
Step 3: Put files in place

First, set file defaults: all of these files should be owned by root and only writable by root (or the Administrators user, on Windows). The puppet_enterprise module has helpful variables for setting these:

```
File {
    owner => $puppet_enterprise::params::root_user,
    group => $puppet_enterprise::params::root_group,
    mode  => $puppet_enterprise::params::root_mode,
}
```

Next, put all relevant plugin files into place, using the $plugin_basedir variable we set above:

```
file {"${plugin_basedir}/agent/nrpe.ddl":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/agent/nrpe.ddl',
}

file {"${plugin_basedir}/agent/nrpe.rb":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/agent/nrpe.rb',
}
```

(Optional) Step 4: Configure the plugin

Some agent plugins require extra configuration to work properly. If present, these settings must be present on every agent node that will be using the plugin.

The main server.cfg file is managed by the mcollective class in the puppet_enterprise module. Although editing it is possible, it is not supported. Instead, you should take advantage of the MCollective daemon's plugin config directory, which is located at "${mco_etc}/plugin.d".

- File names in this directory should be of the format <agent name>.cfg.
- Setting names in plugin config files are slightly different:

<table>
<thead>
<tr>
<th>In server.cfg</th>
<th>In ${mco_etc}/plugin.d/nrpe.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin.nrpe.conf_dir = /etc/nagios/</td>
<td>conf_dir = /etc/nagios/nrpe</td>
</tr>
<tr>
<td>nrpe</td>
<td></td>
</tr>
</tbody>
</table>

You can use a normal file resource to create these config files with the appropriate values. For simple configs, you can set the content directly in the manifest; for complex ones, you can use a template.

```
file {"${mco_etc}/plugin.d/nrpe.cfg":
    ensure => file,
    content => "conf_dir = /etc/nagios/nrpe\n",}
```

Distributing Policy files

You can also distribute policy files for the ActionPolicy authorization plugin. This can be a useful way to completely disable certain unused actions, limit actions so they can only be used on a subset of your agent nodes, or allow certain actions from the command line.

These files should be named for the agent plugin they apply to, and should go in ${mco_etc}/policies/<plugin name>.cfg. Policy files should be distributed to every agent node that runs the plugin you are configuring.
Note: The policies directory doesn’t exist by default; you will need to use a file resource with ensure => directory to initialize it.

The policy file format is documented here. When configuring caller IDs in policy files, note that PE uses the following ID by default:
cert=peadmin-public — the command line client, as used by the peadmin user on the Puppet master server.

Example: This code would completely disable the package plugin’s update option, to force users to do package upgrades through your centralized Puppet code:

```puppet
file {
  
  "$mco_etc/policies": ensure => directory,
}

file {
  "$mco_etc/policies/package.policy":
  ensure  => file,
  content => "policy default allow
deny * update * *",
}
```

Note: Additionally, you can use the puppet_enterprise::profile::mcollective::agent::allowed_actions parameter to create policy files. Use allowed_actions to specify agent plugins you want to apply an action policy to, and a list of the actions you want to explicitly allow. If a plugin is not specified, no policy is created for it. The behavior without a policy is determined by the allow_no_actionpolicy parameter. We've also included a default policy for the package plugin that disables install, uninstall, and purge.

Step 5: Assign the class to nodes

For plugins you are distributing to all agent nodes, you can use the console to assign your class to the special PE MCollective group. (This group is automatically maintained by the console, and contains all nodes which have not been added to the special no mcollective group.)

For plugins you are only distributing to some agent nodes, you must do the following:

- Create two Puppet classes for the plugin: a main class that installs everything, and a "client" class that only installs the .ddl file and the supporting plugins.
- Assign the main class to any agent nodes that should be running the plugin.
- Assign the "client" class to the PE Console and PE Master groups in the console. (These special groups contain all of the console and Puppet master nodes in your deployment, respectively.)

Step 6: Run Puppet and confirm the plugin is installed

You can either wait for the next scheduled run, or trigger an on-demand run.

Follow the instructions in the MCollective documentation to verify that your new plugins are properly installed.

Example plugin class

This is an example of a class that installs the nrpe plugin.

The files directory of the module would simply contain a complete copy of the nrpe plugin’s Git repo. In this example, we are not creating separate agent and client classes.

```puppet
# /etc/puppetlabs/code/environments/production/modules/mco_plugins/
manifests/nrpe.pp

class mco_plugins::nrpe {   
  Class['puppet_enterprise::mcollective::server::plugins'] -> Class[$title] 
  ~> Service['mcollective']   
  include puppet_enterprise::params 
  $plugin_basedir = "${puppet_enterprise::params::mco_plugin_userdir}/mcollective"
```

© 2020 Puppet, Inc.
$mco_etc = $puppet_enterprise::params::mco_etc

File {
    owner => $puppet_enterprise::params::root_user,
    group => $puppet_enterprise::params::root_group,
    mode  => $puppet_enterprise::params::root_mode,
}

file "${plugin_basedir}/agent/nrpe.ddl":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/agent/nrpe.ddl",
}

file "${plugin_basedir}/agent/nrpe.rb":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/agent/nrpe.rb",
}

file "${plugin_basedir}/aggregate/nagios_states.rb":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/aggregate/nagios_states.rb",
}

file "${plugin_basedir}/application/nrpe.rb":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/application/nrpe.rb",
}

file "${plugin_basedir}/data/nrpe_data.ddl":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/data/nrpe_data.ddl",
}

file "${plugin_basedir}/data/nrpe_data.rb":
    ensure => file,
    source => 'puppet:///modules/mco_plugins/mcollective-nrpe-agent/data/nrpe_data.rb",
}

# Set config: If this setting were in the usual server.cfg file, its name would
# be plugin.nrpe.conf_dir
file "${mco_etc}/plugin.d/nrpe.cfg":
    ensure => file,
    content => "conf_dir = /etc/nagios/nrpe",
}


---

Disabling MCollective

Turn off MCollective on a single node or on all nodes.

Related information

Move from MCollective to Puppet orchestrator on page 751
Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

### Removing MCollective on page 752
Remove MCollective and its related files from the nodes in your infrastructure.

### Disable MCollective on select nodes
Since the Puppet master server supports managing non-PE agent nodes (including things like network devices), you should disable MCollective on these nodes.

To disable MCollective for a node, in the console, create a rule in the PE MCollective group that excludes the node. This will prevent PE from attempting to enable MCollective on that node.

**Related information**
- [Add nodes to a node group on page 367](#)
  - There are two ways to add nodes to a node group.

### Disable MCollective on all nodes
Turn off MCollective and stop mco commands from running on your system.

1. In the console, click **Classification**, and select the node group **PE Infrastructure**.
2. On the **Configuration** tab, find the puppet-enterprise class. Select the **mcollective** parameter and edit its value to **stopped**.

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet-enterprise</td>
<td>mcollective</td>
<td>stopped</td>
</tr>
</tbody>
</table>

3. Click **Add parameter** and commit the change.
4. Set up a job and run on the **PE Agent (production)** node group to enforce your changes.

mco commands will no longer run on the nodes in your infrastructure.

### Change the port used by MCollective/ActiveMQ
You can change the port that MCollective/ActiveMQ uses with a simple variable change in the console.

1. In the console, click **Classification**, and in the **PE infrastructure** group, select the **PE MCollective group**.
2. On the **Variables** tab, specify a variable:
   - **key** -- Enter **fact_stomp_port**.
   - **value** -- Enter the port number you want to use.
3. Click **Add variable** and commit changes.

### Moving from MCollective to Puppet orchestrator
Puppet Enterprise 2018.1 is the last release to support MCollective. To prepare for these changes, migrate your MCollective work to Puppet orchestrator to automate tasks and create consistent, repeatable administrative processes.

A benefit of moving to Puppet orchestrator is that its architecture requires less infrastructure than MCollective. These diagrams illustrate how distributing the Puppet agent workload with additional compile masters also requires additional ActiveMQ Brokers in an MCollective environment.

© 2020 Puppet, Inc.
Move from MCollective to Puppet orchestrator

Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

Before you begin

Compare your current MCollective workflows to the corresponding features of PE.

<table>
<thead>
<tr>
<th>Feature</th>
<th>In MCollective</th>
<th>In Puppet Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy configuration changes and enforce the state of selected nodes.</td>
<td>Commands <code>mco puppet runonce</code> and <code>mco puppet runall</code> from <code>mcollective-package-agent</code></td>
<td>Running Puppet on nodes on page 364</td>
</tr>
<tr>
<td>Install, uninstall, update, and check the status of packages on a node.</td>
<td>Command <code>mco package *</code> from <code>mcollective-package-agent</code></td>
<td>puppetlabs-package module that installs with PE</td>
</tr>
<tr>
<td>Start, stop, and check the status of services running on a node.</td>
<td>Command <code>mco service *</code> from <code>mcollective-service-agent</code></td>
<td>puppetlabs-service module that installs with PE</td>
</tr>
<tr>
<td>Limit user access to tasks that run on selected nodes.</td>
<td><code>mcollective-actionpolicy-auth</code></td>
<td>PE role-based access control (RBAC)</td>
</tr>
</tbody>
</table>
### Removing MCollective

Remove MCollective and its related files from the nodes in your infrastructure.

**Note:** This procedure does not remove your MCollective or ActiveMQ log files.

1. In the console, click **Classification**, and select the node group **PE Infrastructure**.
2. On the **Configuration** tab, find the **puppet_enterprise** class. Select the **mcollective** parameter and edit its value to **absent**.

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>puppet_enterprise</td>
<td>mcollective</td>
<td>absent</td>
</tr>
</tbody>
</table>

3. Click **Add parameter** and commit the change.
4. Set up a job and run on the **PE Agent (production)** node group to enforce your changes.
The server components of, including pe-activemq and the peadmin user, are removed from the master and the service on agents is stopped. You must upgrade to 2019.0 or later to completely remove from agents.

**Related information**

Move from MCollective to Puppet orchestrator on page 751

Move your MCollective workflows to orchestrator and take advantage of its integration with Puppet Enterprise console and commands, APIs, role-based access control, and event tracking.

## Maintenance

- **Backing up and restoring Puppet Enterprise** on page 753
  
  Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

- **Puppet Enterprise database maintenance** on page 763
  
  You can optimize the Puppet Enterprise (PE) databases to improve performance.

### Backing up and restoring Puppet Enterprise

Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new master, troubleshoot, and quickly recover in the case of system failures.

**Note:** Split installations, where the master, console, and PuppetDB are installed on separate nodes, are deprecated in later versions of PE. We recommend migrating from an existing split installation to a monolithic installation—with or without compilers—and a standalone PE-PostgreSQL node. See the docs for how to Migrate from a split to a monolithic installation on page 228

**Important:** Always store your backups in a safe location that you can access if your master fails. Backup files are not encrypted, so secure these as you would any sensitive information.

You can use the `puppet-backup` command to back up and restore your primary master or master of masters in monolithic installations only. You can't use this command to back up split installations or compile masters. 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 master, uninstall and reinstall PE before restoring your infrastructure. If you are restoring or migrating your infrastructure to a master with a different hostname than the previous master, you'll redirect your agents to the new master during the restore process. In both cases, the freshly installed PE must be the same PE version that was in use when you backed up the files.

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.

- Backup and restore split installations and upgrades on page 754
- Backup and restore monolithic installations and same-version infrastructure on page 758

**Backup and restore split installations and upgrades**

Backup and restore your Puppet Enterprise (PE) using this method if you are on a split installation or if you are migrating to a new version of PE.

**Back up your database and Puppet Enterprise files**

To properly back up your PE installation, you'll need to back up certain files and databases.

1. Back up these files:

   - `/etc/puppetlabs/`---if you back up this directory to `/opt/puppetlabs/server/data/postgres/backups`, you can later access your original installation answer file or `pe.conf` file when you restore. If you're restoring from a version earlier than 2016.2, you need to convert your answer file to the `pe.conf` file format.
   - The modulepath---if you've configured it to be outside the PE default of `modulepath = /etc/puppetlabs/code/environments/production/modules:/etc/puppetlabs/code/modules:/opt/puppetlabs/puppet/modules` in `puppet.conf`
   - Ensure any SSH keys used for Code Manager are backed up. This key is stored in `/etc/puppetlabs/puppetserver/conf.d/code-manager.conf` in the `private-key` setting. If the file does not exist, Code Manager is not enabled, and this step is not needed.
2. **Back up your PE PostgreSQL database.**

   **Note:** Run all back up and restore commands as the pe-postgres user, which must have write access to the backup location to create the backup, and read access to it to restore it.

   On a split install (master, console, PuppetDB/PostgreSQL each on a separate node), they will be located across the various servers assigned to these PE components.

   • `/etc/puppetlabs/`: different versions of this directory can be found on the server assigned to the Puppet master component, the server assigned to the console component, and the server assigned to the PuppetDB/PostgreSQL component. You should back up each version.
   
   • `/opt/puppetlabs/server/data/console-services/certs/`: located on the server assigned to the console component.
   
   • `/opt/puppetlabs/server/data/postgresql/9.6/data/certs/`: located on the server assigned the PuppetDB
   
   • The pe-activity, pe-classifier, pe-orchestrator, pe-puppetdb, and pe-rbac databases: located on the server assigned to the PuppetDB/PostgreSQL component.

   To back up each database individually, run the following commands:

   ```
sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump -Fc pe-activity -f /tmp/pe-activity_`date +%m_%d_%y_%H_%M`.bin
sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump -Fc pe-rbac -f /tmp/pe-rbac_`date +%m_%d_%y_%H_%M`.bin
sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump -Fc pe-classifier -f /tmp/pe-classifier_`date +%m_%d_%y_%H_%M`.bin
sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump -Fc pe-puppetdb -f /tmp/pe-puppetdb_`date +%m_%d_%y_%H_%M`.bin
sudo -u pe-postgres /opt/puppetlabs/server/bin/pg_dump -Fc pe-orchestrator -f /tmp/pe-orchestrator_`date +%m_%d_%y_%H_%M`.bin
```

   **Tip:** If you are not in a directory that the pe-postgres user has permissions to access, you'll receive a 'permission denied' error, even though the pg_dump command will succeed.

---

**Restore your database and Puppet Enterprise files**

Restore your database and Puppet Enterprise files.

1. **Using the standard install process** (run the puppet-enterprise-installer script), reinstall the same version of Puppet Enterprise that was installed for the files you backed up.

   • If you're restoring from a version earlier than 2016.2, you need to covert your answer file to the pe.conf file format, and use that during the installation process.
   
   • If you need to review the PE installation process, check out Installing Puppet Enterprise.
2. Stop all PE services, except `pe-postgresql`.
   a. On the Puppet master, run:

```
puppet resource service puppet ensure=stopped
puppet resource service pe-puppetserver ensure=stopped
puppet resource service pe-activemq ensure=stopped
puppet resource service mcollective ensure=stopped
puppet resource service pe-orchestration-services ensure=stopped
puppet resource service pe-nginx ensure=stopped
```

b. Run both commands on the node assigned the PuppetDB/PostgreSQL component:

```
puppet resource service pe-puppetdb ensure=stopped
puppet resource service puppet ensure=stopped
```

c. Run both commands on the node assigned the console component:

```
puppet resource service pe-console-services ensure=stopped
puppet resource service puppet ensure=stopped
```

3. Restore your PE PostgreSQL databases. Depending on the size of these databases, restoration might take a considerable amount of time to complete.

   To restore an individual database backup, run the following command: `sudo -u pe-postgres /opt/puppetlabs/server/apps/postgresql/bin/pg_restore -Cc -d template1 <BACKUP_FILE>.bin` This command connects to the `template1` database and then drops the database indicated in `<BACKUP_FILE>.bin`. The command then re-creates that database and connects to the newly created database to perform the restoration.

   If you omit the `-d` parameter, `pg_restore` will output the SQL commands needed to restore a database, but it will not perform the actual restoration.

4. Stop the `pe-postgresql` service:

```
puppet resource service pe-postgresql ensure=stopped
```
5. From your /etc/puppetlabs/ backup, restore these directories and files:
   - /etc/puppetlabs/puppet/puppet.conf
   - /etc/puppetlabs/puppet/ssl (fully replace with backup, do not leave existing ssl data)
   - /etc/puppetlabs/puppetdb/ssl (fully replace with backup, do not leave existing ssl data)
   - /opt/puppetlabs/server/data/postgresql/9.6/data/certs/
   - /opt/puppetlabs/server/data/console-services/certs/
   - The modulepath—if you’ve configured it to be something other than the PE default.

These files and databases should be replaced on the various servers assigned to these PE components.

   - /etc/puppetlabs/: as noted earlier, there is a different version of this directory for the Puppet master component, the console component, and the database support component (i.e., PuppetDB and PostgreSQL). You should replace each version.
   - /etc/puppetlabs/puppetdb/ssl: located on the server assigned to the PuppetDB component.
   - /opt/puppetlabs/server/data/console-services/certs/: located on the server assigned to the console component.
   - /opt/puppetlabs/server/data/postgresql/9.6/data/certs/: located on the server assigned the PuppetDB/ PostgreSQL component.
   - The pe-classifier, pe-rbac, and pe-activity databases: located on the server assigned to the database support component.
   - The PuppetDB database: located on the server assigned to the database support component.
   - The modulepath: located on the server assigned to assigned to the Puppet master component.

If you backed up any Simple RPC agents, you will need to restore these on the same server assigned to the Puppet master component.

6. Restore modules, manifests, Hiera data, and, if necessary, Code Manager SSH keys. These are typically located in the /etc/puppetlabs/ directory, but you may have configured them in another location.

7. Remove the cached catalog on the Puppet master: `rm -f /opt/puppetlabs/puppet/cache/client_data/catalog/<CERTNAME>.json`

8. Run the following commands:

```
chown pe-puppet:pe-puppet /etc/puppetlabs/puppet/puppet.conf
chown -R pe-puppet:pe-puppet /etc/puppetlabs/puppet/ssl/
chown -R pe-console-services /opt/puppetlabs/server/data/console-services/certs/
chown -R pe-orchestration-services:pe-orchestration-services /etc/puppetlabs/orchestration-services/ssl/
chown -R pe-postgres:pe-postgres /opt/puppetlabs/server/data/postgresql/9.6/data/certs/
chown -R pe-puppetdb:pe-puppetdb /etc/puppetlabs/puppetdb/ssl/
```
9. Restart PE services.
   
a. On the Puppet master, run:

   ```bash
   puppet resource service pe-puppetserver ensure=running
   puppet resource service pe-orchestration-services ensure=running
   puppet resource service pe-activemq ensure=running
   puppet resource service pe-nginx ensure=running
   ```

   b. Run the following commands on the node assigned the PuppetDB/PostgreSQL component:

   ```bash
   puppet resource service pe-postgresql ensure=running
   puppet resource service pe-puppetdb ensure=running
   ```

   c. Run the following on the node assigned the console component: puppet resource service pe-console-services ensure=running

   d. Run the following on the Puppet master, PuppetDB, and console components: puppet resource service puppet ensure=running

10. Fix database access privileges with Puppet. Run the following command: puppet infrastructure configure.

**Purge the Puppet Enterprise installation (optional)**

If you're planning on restoring your databases and PE files to the same server(s), you'll want to first fully purge your existing Puppet Enterprise installation.

1. You'll need to run the uninstaller script `puppet-enterprise-uninstaller` with the `-p` and `-d` flags on each server that has been assigned a component. It can be found either at `/opt/puppetlabs/bin/puppet-enterprise-uninstaller` or inside the installer tarball you downloaded when setting up PE initially.

2. After running the uninstaller, ensure that `/opt/puppetlabs/` and `/etc/puppetlabs/` are no longer present on the system.

For more information about using the PE uninstaller, refer to Uninstalling Puppet Enterprise.

**Backup and restore monolithic installations and same-version infrastructure**

Use this backup and restore method only if you are on a monolithic install and are not upgrading your version of PE.

**Back up your PE infrastructure**

PE backup creates a copy of your Puppet infrastructure, including configuration, certificates, code, and PuppetDB.

**Before you begin**

You can back up monolithic masters only, whether they are the only master or master of masters. The backup and restore commands are not supported for split installations or compile masters.

On your primary master, from the command line logged in as root, run `puppet-backup create`.

To change the default command behavior, pass option flags and values to the command. See the backup and restore reference for a complete list of options.

For example, to limit a backup to certain parts of your PE infrastructure, pass the `--scope` option, specifying the scopes in a comma-separated list. To change the name of your backup file, pass the `--name` option with a string specifying the filename. For example, to back up PuppetDB only and name your file with the scope and date, run:

```bash
puppet-backup create --scope=puppetdb --name=puppetdb_backup_03032018.tgz
```

By default, if you don't specify the `--dir` or `--name` options, PE creates files to `/var/puppetlabs/backups` and names them with a timestamp, such as `pe_backup-<TIMESTAMP>.tgz`.

After backing up, you can move your backup files to another location. Always store your backups in a safe location that is not on the master.
**Related information**

**Configuration parameters and the `pe.conf` file** on page 178

A `pe.conf` file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

**Restore your Puppet Enterprise infrastructure**

Use the restore commands to migrate your PE master to a new host or to recover from system failure.

**Before you begin**

Remember that you must restore files to a fresh installation of the same version of PE used in your backup file.

1. If you are restoring to an existing master, purge any existing PE installation from it.
   a) On the master, uninstall PE by running:
   ```bash
   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.
   For details about uninstalling PE, see the uninstalling documentation.

2. Install PE on the master you are restoring to. This must be the same PE version that was used for 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:
   ```bash
   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:
   ```bash
   sudo ./puppet-enterprise-installer
   ```
   For details about the PE installation process, see the installing documentation.

3. On the master logged in as root, restore your PE infrastructure by running:
   ```bash
   puppet-backup restore <BACKUP-FILENAME>
   ```
   To change the default command behavior, pass option flags and values to the command. See the command reference for a complete list of options.
   For example, to restore your scope, certificates, and code from one backup file, but your PuppetDB data from another, pass the `--scope` option:
   ```bash
   puppet-backup restore pe_backup-2018-03-03_20.07.15_UTC.tgz --scope=config,certs,code
   ```
   ```bash
   puppet-backup restore pe_backup-2018-04-04_20.07.15_UTC.tgz --scope=puppetdb
   ```

4. If you restored PE onto a master 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 master. 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 master by running:
   ```bash
   bolt task run puppet_conf action=set section=agent setting=server value=<RESTORE_HOSTNAME> --nodes <COMMA-SEPARATED LIST OF NODES>
   ```
   ```bash
   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 master **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 master.
5. 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

Related information
Configuration parameters and the pe.conf file on page 178
A pe.conf file is a HOCON formatted file that declares parameters and values needed to install, upgrade, and configure PE.

Installing Puppet Enterprise on page 172
You can install PE in a monolithic configuration, where all infrastructure components are installed on one node, or in a split configuration, where the master, PuppetDB, and console are installed on separate nodes.

Uninstalling on page 221
Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

Backup and restore reference
Use these options to change the backup and restore scope and other options for the puppet-backup command.

Note: Backup commands must be run as root.

puppet-backup create
Run the puppet-backup create command to create backup files of your PE infrastructure.

Usage:

    puppet-backup create [--dir=<DIRECTORY_PATH>] [--name=<BACKUP_NAME>.tgz] [--scope=<SCOPE_LIST>] [--force]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>--dir=BACKUP_DIR</td>
<td>Specifies the directory to write the backup file to.</td>
<td>A valid filepath that the pe-postgres user has write permission for.</td>
<td>/var/puppetlabs/backups/</td>
</tr>
<tr>
<td>--name=BACKUP_NAME.tgz</td>
<td>Specifies the name for the backup file.</td>
<td>A string designating a file name.</td>
<td>pe_backup-&lt;TIMESTAMP&gt;.tgz</td>
</tr>
<tr>
<td>--pe-environment=ENVIRONMENT</td>
<td>Specifies the environment to back up. To ensure configuration is recovered correctly, this should be the environment where your master is located.</td>
<td>A valid environment name.</td>
<td>production</td>
</tr>
<tr>
<td>--scope=SCOPE</td>
<td>Scope of backup to create.</td>
<td>Either all or any combination of the other available scopes, as a comma-separated list: • config: PE configuration including license, classification, and RBAC settings. Does not include</td>
<td>all</td>
</tr>
</tbody>
</table>

© 2020 Puppet, Inc.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>--force</td>
<td>Bypass validation checks and ignore warnings.</td>
<td>None.</td>
<td>If you don't specify --force, PE verifies that the destination directory exists and has enough space for the backup process.</td>
</tr>
</tbody>
</table>

For example, to create a backup of PuppetDB only and give it a name that shows the scope of the file:

```
puppet-backup create --scope=puppetdb --name=puppetdb_backup_03032018.tgz
```

**puppet-backup restore**

Run the `puppet-backup restore` command to restore your PE infrastructure from backup files.

Usage:

```
puppet-backup restore <PATH/TO/BACKUP_FILE.tgz> [--scope=<SCOPE_LIST>] [--force]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>--pe-environment=ENVIRONMENT</td>
<td>Specifies the environment to restore.</td>
<td>A valid environment name for which you have an existing backup.</td>
<td>production</td>
</tr>
</tbody>
</table>
| --scope=SCOPE               | Scope of backup to restore. All scopes must eventually be restored, but you can restore different scopes from different backup files with successive restore commands. | Either all or any combination of the other available scopes, as a comma-separated list:  
  - `config`: PE configuration including license, classification, and RBAC settings. Does not include Puppet gems or Puppet Server gems.  
  - `certs`: PE CA certificates and full SSL directory | all                      |
### Option Description Values Default

- **code**: Puppet code deployed to your codedir at backup time.
- **puppetdb**: PuppetDB data, including facts, catalogs, and historical reports
- **all**: All listed scopes.

---

**--force**

Bypass validation checks and ignore warnings.

None.

If you don't specify **--force**, PE verifies that the destination directory exists and has enough space for the restore process. Returns warnings for insufficient space or invalid locations.

---

For example, to restore PuppetDB from one backup file and restore configuration, certificates, and code from another backup file:

```
puppet-backup restore /mybackups/pe_backup_03032018.tgz --scope=puppetdb

puppet-backup restore /mybackups/pe_backup_04042018.tgz --scope=config,certs,code
```

### Directories and data backed up

<table>
<thead>
<tr>
<th>Scope</th>
<th>Directories and databases backed up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>certs</strong></td>
<td>• /etc/puppetlabs/puppet/ssl/</td>
</tr>
</tbody>
</table>
| **code** | • /etc/puppetlabs/code/  
• /etc/puppetlabs/code-staging/  
• /opt/puppetlabs/server/data/puppetserver/filesync/storage/ |
| **config** | • Orchestrator database  
• RBAC database  
• Classifier database  
• /etc/puppetlabs/.except:  
• /etc/puppetlabs/code/  
• /etc/puppetlabs/code-staging/  
• /etc/puppetlabs/puppet/ssl |
### Puppet Enterprise database maintenance

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

- Databases in Puppet Enterprise on page 763
  PE uses PostgreSQL as the backend for its databases. Use the native tools in PostgreSQL to perform database exports and imports.
  
- Optimize a database on page 764
  If your databases are slow, begin taking up too much disk space, or need general performance enhancements, use the PostgreSQL `vacuum` command to optimize any of the PE databases.

- List all database names on page 764
  Use these instructions to list all database names.

### 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>
</tbody>
</table>

---

© 2020 Puppet, Inc.
Optimize a database

If your databases are slow, begin taking up too much disk space, or need general performance enhancements, use the PostgreSQL vacuum command to optimize any of the PE databases.

To optimize a database, run: `su - pe-postgres -s /bin/bash -c "/opt/puppetlabs/server/apps/postgresql/bin/vacuumdb -z --verbose <DATABASE NAME>"`

Related information

List all database names on page 764
Use these instructions to list all database names.

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:
   
   ```bash
   /opt/puppetlabs/server/bin/psql
   ```

3. List the databases:
   
   ```bash
   \l
   ```

4. Exit the PostgreSQL command line:
   
   ```bash
   \q
   ```

5. Log out of the pe-postgres user:
   
   ```bash
   logout
   ```

Troubleshooting

Use this guide to troubleshoot issues with your Puppet Enterprise installation.

- **Troubleshooting the installer** on page 765
  If the installer fails, check for configuration or installation issues.
- **Troubleshooting connections between components** on page 765
  If agent nodes can’t retrieve configurations, check for communication, certificate, DNS, and NTP issues.
- **Troubleshooting Code Manager**
Troubleshooting the databases on page 767
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 MCollective on page 768
If you have performance issues with the ActiveMQ service, you can increase the ulimit or use the ActiveMQ console to get troubleshooting information.

Troubleshooting Windows on page 769
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 the installer
If the installer fails, check for configuration or installation 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 Puppet agents can reach the Puppet master hostname you chose during installation.
2. Verify that the Puppet master can reach itself at the Puppet master hostname you chose during installation.
3. If the master 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. Verify that inbound traffic is allowed on required ports.
   
   If you installed the Puppet master and the console on the same server, it must accept inbound traffic on ports 8140, 61613, and 443.
   
   If you installed the master and the console on different servers, the master must accept inbound traffic on ports 8140 and 61613 and the console must accept inbound traffic on ports 8140 and 443.

2. If your Puppet master has multiple network interfaces, verify that the master allows traffic via the IP address that its valid DNS names resolve to, not just via an internal interface.

The console was installed before the Puppet master
If you are installing the console and the Puppet master on separate servers, you must install the console first.

Troubleshooting connections between components
If agent nodes can't retrieve configurations, check for communication, certificate, DNS, and NTP issues.

Agents can't reach the Puppet master
Agent nodes must be able to communicate with the Puppet master in order to retrieve configurations.

If agents can't reach the Puppet master, running telnet <puppet master's hostname> 8140 returns the error "Name or service not known."

1. Verify that the Puppet master 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 Puppet master.

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 master, view a list of pending certificate requests: puppet cert list
2. Sign a specified node's certificate: puppet cert sign <NODE NAME>

Agents aren't using the master's valid DNS name
Agents trust the master only if they contact it at one of the valid hostnames specified when the master 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 master, the node and master can't establish communication.

1. To edit the master's hostname on nodes, in /etc/puppetlabs/puppet/puppet.conf, change the server setting to a valid DNS name.
2. To reset the master's valid DNS names, run:

   `/etc/init.d/pe-nginx stop
   puppet cert clean <MASTER_CERTNAME>
puppet cert generate <MASTER_CERTNAME> --dns_alt_names=<COMMA-SEPARATED_LIST_OF_DNS_NAMES>
   /etc/init.d/pe-nginx start`

Time is out of sync
The date and time must be in sync on the Puppet master 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 master --configprint ssldir)/certs/<NODE NAME>.pem` returns invalid dates, such as certificates dated in the future.

1. On the master, delete certificates with invalid dates: puppet cert clean <NODE NAME>
2. On nodes with invalid certificates, delete the SSL directory: `rm -r $(puppet agent --configprint ssldir)`
3. On agent nodes, generate a new certificate request: puppet agent --test
4. On the master, sign the request: puppet cert sign <NODE NAME>

A node is re-using a certname
If a node re-uses an old node's certname and the master retains the previous node's certificate, the new node is unable to request a new certificate.

1. On the master, clear the node's certificate: puppet cert clean <NODE NAME>
2. On agent node, generate a new certificate request: puppet agent --test
3. On the master, sign the request: puppet cert sign <NODE NAME>
Agents can't reach the filebucket server

If the master is installed with a certname that doesn't match its hostname, agents can't back up files to the filebucket on the Puppet master.

If agents log errors like "could not back up," nodes are likely attempting to back up files to the wrong hostname.

On the master, edit /etc/puppetlabs/code/environments/production/manifests/site.pp so that filebucket server attribute points to the correct hostname:

```puppet
# Define filebucket 'main':
filebucket { 'main':
  server => '<PUPPET_MASTER_DNS_NAME>',
  path   => false,
}
```

Changing the filebucket server attribute on the master fixes the error on all agent nodes.

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.

**Related information**

- Install using text mode (mono configuration) on page 174
- Install using text mode (split configuration) on page 174

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 master, 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:

   ```bash
   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 may 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 master 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 MCollective

If you have performance issues with the ActiveMQ service, you can increase the ulimit or use the ActiveMQ console to get troubleshooting information.

ActiveMQ generates errors about too many open files or not enough memory

The ulimit must be adequate for the number of files (`nofile`) and number of processes (`nproc`) handled in your environment.

To increase the ulimit for the `pe-activemq` user, edit `/etc/security/limits.conf` so that it contains:

```
pe-activemq   soft     nproc  8192
pe-activemq   hard     nproc  8192
pe-activemq   soft     nofile 16384
pe-activemq   hard     nofile 16384
```

ActiveMQ doesn’t perform as expected

The ActiveMQ service provides a console that can be used for advanced debugging, such as pulling performance metrics from the AMQ service while it's running. The console must be configured for access.

1. On the Puppet master, edit `/etc/puppetlabs/activemq/jetty.xml` so that the "host" value of the "connectors" property is set to "0.0.0.0". For example:

   ```xml
   <bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">
   <!-- the default port number for the web console -->
   <property name="host" value="0.0.0.0"/>
   <property name="port" value="8161"/>
   </bean>
   ```

2. In the console, click Classification, and in the PE Infrastructure node group, select the PE ActiveMQ Broker node group.

3. On the Configuration tab, under the `puppet_enterprise::profile::amq::broker` class, set the `enable_web_console` parameter to true.

4. Click Add parameter and commit changes.

5. Run Puppet on the agents in the PE ActiveMQ Broker group.
Access the ActiveMQ console from the node with the puppet_enterprise::profile::amq::broker class on port 8161 with /admin, for example, http://hostname:8161/admin. The password and username for the ActiveMQ console is defined at /etc/puppetlabs/activemq/jetty-realm.properties.

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 Puppet master 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 master hostname, Puppet applies the new value in $confdir\puppet.conf.

When you upgrade Windows, you must use the same master hostname that you specified when you installed.

For information on what settings are preserved during an upgrade, see installing Puppet agent on Windows.

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 must be escaped. For example, these are valid file resources:

```plaintext
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:

```plaintext
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`:

```puppet
evac { '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:

```puppet
evac { '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`. © 2020 Puppet, Inc.
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 may 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 -addinstore 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 Puppet master that cannot be executed.

  Set the user/group executable bits appropriately on the master:

  ```erb
  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 master, 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.

Messages from the Marionette Collective Server, Puppet PXP Agent services, and the Puppet Agent, can be viewed in
the Windows Event Viewer (Windows Logs > Application).

**Note:** There are additional log files located on disk. Marionette Collective produces logs at: `C:\ProgramData \PuppetLabs\mcollective\var\log\mcollective.log`, PXP Agent produces logs at: `C:\ProgramData\PuppetLabs\pxp-agent\var\log\pxp-agent.log`. Puppet can (optionally) be
configured to produce additional logs at: `C:\ProgramData\PuppetLabs\puppet\var\log\puppet.log`.

Enable Puppet to emit `--debug` and `--trace` messages to the Windows Event Log by stopping the Puppet service
and restarting it:

```bash
c:\>sc stop puppet && sc start puppet --debug --trace
```

Note that this setting only takes effect until the next time the service is restarted, or until the system is rebooted.

For more information on logging, see [Configuring Puppet agent on Windows](#).