Continuous Delivery

Contents

Welcome to Continuous Delivery for Puppet Enterprise

Continuous Delivery for Puppet Enterprise (PE) is a tool for streamlining and simplifying the continuous integration and continuous delivery of your Puppet code. Continuous Delivery for PE offers a prescriptive workflow to test and deploy Puppet code across environments.

To harness the full power of Puppet Enterprise, you need a robust system for testing and deploying your Puppet code. Continuous Delivery for PE offers prescriptive, customizable workflows and intuitive tools for Puppet code testing, deployment, and impact analysis, helping you ship changes and additions with speed and confidence.

Note: Continuous Delivery for PE version 4.x is now available! To upgrade to the Continuous Delivery for PE 4.x series from a version in the 3.x series, see Migrating 3.x data to 4.x.

Continuous Delivery for PE docs links	Other useful places
Before you install Release notes Known issues System requirements **Install Continuous Delivery for PE** Install using the cd4pe module **Configure and manage access** Configuring - Integrate with source control and PE, set up job hardware, and configure impact analysis. User permissions and groups Creating and managing workspaces **Learn the basics** Getting started Key concepts **Continuously deliver Puppet code** See the potential impact of new code Create a job to test code Choose a deployment policy	**Docs for related Puppet products** Puppet Enterprise Open source Puppet Puppet Development Kit **Why and how people are using Continuous Delivery for PE** Read our ebook on Critical Considerations for Continuous Delivery **Get support** Search the Support portal and knowledge base Upgrade your support plan **Share and contribute** Engage with the Puppet community Puppet Forge Open source projects from Puppet on GitHub

To send us documentation feedback or let us know about a docs error, use the feedback form at the bottom of each page. Thanks!

Release notes

New features, enhancements, known issues, and resolved issues for the Continuous Delivery for Puppet Enterprise 3.x release series.

- Continuous Delivery for PE release notes on page 5

These are the new features, enhancements, resolved issues, and deprecations for the Continuous Delivery for Puppet Enterprise (PE) 3.x release series.

These are the known issues for the Continuous Delivery for PE 3.x release series.

Continuous Delivery for PE release notes

These are the new features, enhancements, resolved issues, and deprecations for the Continuous Delivery for Puppet Enterprise (PE) 3.x release series.

To upgrade to the Continuous Delivery for PE 4.x series from a version in the 3.x series, go to Migrating 3.x data to 4.x.

Version 3.13.8

Released 1 September 2021

Resolved in this release:

- We've improved the database migration process from versions in the 3.x series to versions in the 4.x series.

Version 3.13.7

Released 26 April 2021

Resolved in this release:

- Impact analysis tasks on modules now manage prefixed environments correctly.

Version 3.13.6

Released 18 March 2021

New in this release:

- **Configure the Bolt PCP read timeout period.** To prevent job run timeouts caused by file sync delays, you can now adjust the Bolt Puppet Communication Protocol (PCP) timeout period by setting the `CD4PE_BOLT_PCP_READ_TIMEOUT_SEC` environment variable. For more information, see Adjusting the timeout period for jobs.

Resolved in this release:

- Impact analysis tasks are now case-insensitive when processing resource names.

Version 3.13.5

Released 17 February 2021

Resolved in this release:

- Jobs now run successfully on pull requests opened from forked copies of source control repositories.

Version 3.13.4

Released 5 November 2020

Resolved in this release:

- Logging for the 3.x to 4.x data migration task is now improved.

Version 3.13.3

Released 27 October 2020

Resolved in this release:

- Code Manager deployments triggered by Continuous Delivery for PE are now automatically retried if certain transient failures occur.

Security notice:

- CVE-2020-25649. A `jackson-databind` vulnerability has been resolved.
- CVE-2020-15250. A JUnit4 vulnerability has been resolved.
- CVE-2020-13956. An Apache HTTPClient vulnerability has been resolved.
- Sonatype-2020-0926. A security scanner may have detected a vulnerability in Continuous Delivery for PE version 3.13.x. However, Continuous Delivery for PE does not exercise the vulnerable code path and is not vulnerable.

Version 3.13.2

Released 27 August 2020

Resolved in this release:

- The 3.x to 4.x migration task now consumes less memory, reducing the possibility of `out of memory` errors.

Version 3.13.1

Released 25 August 2020

Resolved in this release:

- When 3.x data is migrated to 4.x, the database no longer creates duplicate users in the 4.x installation.

Version 3.13.0

Released 25 August 2020

New in this release:

- **Status notification prefixes.** You now have the option to add a custom prefix to the pipeline stage result notifications Continuous Delivery for PE sends to your source control provider. For more information, see Status notification prefixes for source control.
- **Custom deployment policy improvements.** You can now use custom deployment policies when deploying code from your control repo or module repo regex branch pipelines. Additionally, custom deployment policies now support the use of sensitive parameters.

  CAUTION: Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.
- **Data migration to 4.x.** Continuous Delivery for PE 4.x is now available. To upgrade to the Continuous Delivery for PE 4.x series from a version in the 3.x series, see Migrating 3.x data to 4.x.
- **Usability improvements.** Version 3.13.0 introduces several improvements to the design and usability of the web UI, including:
  - More specific error messaging when adding Puppet Enterprise credentials.
  - Clearer controls when setting up a default pipeline.

Resolved in this release:

- Clicking **Message Center** directs you to the messages associated with your username, not the messages associated with your active workspace.
- Deployments using the direct deployment policy now fail with an error if the `max_node_failure` value assigned to the deployment is less than zero.
- Impact analysis tasks run successfully when trace-level logging is enabled.
- Pagination controls on the nodes view of impact analysis reports work as expected.
- Clicking the text of an **Auto promote** control in a pipeline only impacts the associated pipeline stage.
- The functionality of the **User settings** control for the root user is restored.

- When editing a deployment in the web UI after updating a custom deployment policy, the names of custom deployment policies are no longer shown twice.

Version 3.12.4

Released 18 August 2020

New in this release:

- **Export Nodes page information.** You can now generate and export a CSV file of the information in your node table.
- **Support for Bitbucket Server source branch update webhook.** Continuous Delivery for PE now supports the new webhook for source branch updates in a pull request introduced in Bitbucket Server version 7.0.

> **Note:** You must manually configure your Bitbucket Server webhooks to recognize source branch updates in order to use this feature with Continuous Delivery for PE. See the Bitbucket Server documentation for more information.

Resolved in this release:

- The **Nodes** page now displays data for all responsive PE servers if one server is unresponsive.

Version 3.12.3

Released 5 August 2020

Resolved in this release:

- The **Nodes** page now works as expected if SSL is enabled on your Continuous Delivery for PE installation. You must connect using HTTPS.

> **Note:** Firefox users who have enabled SSL and are using a self-signed certificate must add the certificate to the Firefox trust store.

- The **Nodes** page now displays all other available data if any node returns `null` fact data.
- The column selector on the **Nodes** page no longer remains open when you move to a new page of results.
- An issue with cookies has been resolved, and the **Nodes** page now loads correctly when using Safari or Firefox.
- When using PE 2019.7 or 2019.8, clicking **View Report** on the **Nodes** page now correctly directs you to the appropriate report in the PE console.

Version 3.12.2

Released 20 July 2020

Resolved in this release:

- The latest version of the `puppetlabs-cd4pe_jobs` module is no longer required, and jobs in installations where the module has not been upgraded now work as expected.

Version 3.12.1

Released 20 July 2020

Resolved in this release:

- When a super user is deleted from Continuous Delivery for PE, the workspaces the user was a member of now remain intact.

Security notice:

- **CVE-2020-14039.** A security scanner may have detected a Go vulnerability in Continuous Delivery for PE version 3.11.x and 3.12.x. However, Continuous Delivery for PE does not exercise the vulnerable code path and so is not vulnerable.

Version 3.12.0

Released 15 July 2020

New in this release:

- **Maximum concurrent catalog compiles setting.** Each impact analysis-enabled PE instance integrated with Continuous Delivery for PE now has a setting for the maximum number of node catalog compilations each workspace is allowed to perform concurrently – 10 by default. This setting helps to balance the catalog compilation load placed on a PE instance by Continuous Delivery for PE impact analysis report generation. Adjust this setting by editing your PE credentials in the **Settings** area.
- **Customizable job timeout intervals.** You can now adjust the length of time allowed for a job to complete, connect to an endpoint, or access a cached Git repository before timing out. For more information, see Adjusting the timeout period for jobs.
- **Repository caching improvements.** By default, Continuous Delivery for PE now omits the `.git` directory when passing a cached Git repository to job hardware. For more on enabling and customizing Git repository caching, which remains disabled by default, see Improving job performance by caching Git repositories.
- **Usability improvements.** Version 3.12.0 introduces several improvements to the design and usability of the web UI, including:
  - If a node catalog compilation fails, the associated impact analysis task's status is shown as FAILED instead of DONE.
  - Many icons have been updated to offer a cleaner, more streamlined look and feel.

Resolved in this release:

- Custom deployment policies with redundant approval steps – such as those that include a custom approval step and also utilize a built-in deployment policy with its own approval step – now only request approval once.
- Continuous Delivery for PE now only sends a new status update to your source control provider when the pipeline's status has changed.

Security notice:

- **CVE-2020-13692.** A security scanner may have detected a PostgreSQL JBDC driver vulnerability with a 9.8 CVSS score in Continuous Delivery for PE version 3.11.1 and earlier. However, Continuous Delivery for PE does not exercise the vulnerable code path and so is not vulnerable. Continuous Delivery for PE version 3.12.0 includes PostgreSQL version 42.2.13, which resolves the vulnerability.

Version 3.11.1

Released 25 June 2020

Resolved in this release:

- Bitbucket Server users are now able to see all of their organizations, repositories, and branches when adding a control repo or module repo.

Version 3.11.0

Released 24 June 2020

New in this release:

- **Nodes page.** The new **Nodes** page shows information about the nodes from all PE instances integrated with a workspace. You can create a custom table with columns displaying the fact values you're most interested in. To get started using the **Nodes** page, see Reviewing node inventory.

  > **Important:** To use the **Nodes** page, you must upgrade the `puppetlabs-cd4pe` module to version 2.0.1.

- **Control repository webhooks update automatically.** If you update the backend service endpoint for your Continuous Delivery for PE installation, the webhooks connecting the software to your source control provider are automatically updated.

- **Pipeline unique identifier for custom deployment policies.** A new environment variable, `CD4PE_PIPELINE_ID`, is available for inclusion in your custom deployment policies. For more information, see the see the module documentation for `puppetlabs-cd4pe_deployments`.

  CAUTION: Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.

Resolved in this release:

- The temporary branches Continuous Delivery for PE creates during deployments now use only lowercase letters.
- Listing LDAP groups now succeeds when pagination is disabled on your LDAP server.
- Impact analysis task information is now shown correctly when you click **Edit impact analysis** in a pipeline.

Version 3.10.1

Released 16 June 2020

Resolved in this release:

- Git repository caching, which remains disabled by default, is now utilized for deployments.

Version 3.10.0

Released 10 June 2020

New in this release:

- **Clone GitLab repositories using HTTP(S).** GitLab users can now select whether to clone repositories using SSH (default) or HTTP(S). The cloning protocol is set per workspace on the **Source control** page in **Settings**.
- **Improved search for LDAP groups.** This release eliminates a previous limit on the number of LDAP groups recognized by Continuous Delivery for PE, and improves LDAP group search functionality.
- **Usability improvements.** Version 3.10.0 introduces several improvements to the design and usability of the web UI, including:
  - Improved error messaging when an impact analysis task fails to locate Code Manager parameters.

Resolved in this release:

- Duplicate deployment task events are no longer shown on a deployment's details page.
- For users who have enabled Git repository caching, when a symlink is present in a Git repository, Continuous Delivery for PE now copies the symlink instead of its target.
- Jobs that produce a large amount of output no longer deadlock.

Version 3.9.3

Released 5 June 2020

Resolved in this release:

- In order to minimize the unintended effects on smaller repositories, Git repository caching is now disabled by default.

Version 3.9.2

Released 4 June 2020

Resolved in this release:

- The `puppetdb_connection_timeout_sec` class parameter now correctly implements the timeout period you set.

- In some cases, the cached Git repositories that power the job performance improvements introduced in version 3.9.1 can become corrupted. These special cases are now identified and correctly handled by Continuous Delivery for PE when they occur.

Version 3.9.1

Released 2 June 2020

Resolved in this release:

- Users with large Git repositories who are running a Puppet agent on their job hardware no longer experience job timeouts.

Version 3.9.0

Released 28 May 2020

New in this release:

- **Configure PuppetDB timeout for impact analysis tasks.** You can now use the new `puppetdb_connection_timeout_sec` key in your `.cd4pe.yaml` files to set a PuppetDB timeout period for impact analysis tasks.
- **Usability improvements.** Version 3.9.0 introduces several improvements to the design and usability of the web UI, including:
  - A more helpful error message if an invalid URL is entered as an Artifactory endpoint.
  - Updated icons and improved readability on the **Control Repos** and **Modules** pages.

Resolved in this release:

- Impact analysis reports now correctly reflect Hiera data changes in modules.
- Installations that include job hardware running a Puppet agent are now able to promote to the next stage of a module pipeline following a stage that ends with a job.
- Azure DevOps webhook URLs are now parsed correctly when using the deprecated `visualstudio.com` URL.

Version 3.8.0

Released 13 May 2020

New in this release:

- **Deployment approval functions for custom deployment policies.** Two new functions, `approve_deployment` and `decline_deployment` are available for inclusion in your custom deployment policies. For more information, see the see the module documentation for `puppetlabs-cd4pe_deployments`.

  CAUTION: Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.
- **Optional mail attribute setting for LDAP configurations.** When configuring your LDAP integration, you now have the option to select the LDAP user attribute that will identify each member's email address in Continuous Delivery for PE. If unset, this field defaults to `mail`.
- **Environment variables for jobs.** When composing a non-Docker-based job, you can now use the `$REPO_DIR` environment variable to reference the directory that houses the relevant control repo or module repo. Additionally, Continuous Delivery for PE now automatically locates and sets the `$HOME` environment variable before running a job.
- **Usability improvements.** Version 3.8.0 introduces several improvements to the design and usability of the web UI, including:
  - You can now search branch names when selecting the location of a `.cd4pe.yaml` file.

Resolved in this release:

- Entering the bind DN password is no longer required when disabling an LDAP configuration stored in Continuous Delivery for PE.
- The status of impact analysis tasks is now displayed correctly on the impact analysis report's overview page.
- The latest job and deployment activity for each repo is now displayed properly on the **Control Repos** and **Modules** pages.
- The name of the pipeline used in a deployment is now shown correctly in deployment approval request emails and message center messages.
- You can now successfully complete a manual promotion from any stage in a pipeline which ends with a job.
- Users who built module pipelines using an earlier version of Continuous Delivery for PE and then upgraded to the 3.x series now receive the correct URL to the module deployment approval decision page in deployment approval emails.

Version 3.7.1

Released 5 May 2020

Resolved in this release:

- An error and failure of one deployment no longer occurs whenever two deployments are running in the same workspace at the same time.
- Deployments no longer unexpectedly restart when certain conditions are met.

Version 3.7.0

Released 28 April 2020

New in this release:

- **Transfer workspace ownership between users.** Super users and the root user can now reassign ownership of a workspace to a different Continuous Delivery for PE user. For instructions, see Transfer ownership of a workspace.
- **Set user group permissions on a subset of modules.** You can now create user groups that have permissions on only a subset of the modules in your workspace.
- **New direct deployment policy parameter: `fail_if_no_nodes`.** Deployments using the direct deployment policy can now tell Continuous Delivery for PE to stop the deployment and report a failure if the selected environment node group doesn't contain any nodes.
- **Usability improvements.** Version 3.7.0 introduces several improvements to the design and usability of the web UI, including:
  - The email address field used for creating login credentials is now case insensitive.
  - The commit at the HEAD of a feature branch is now automatically selected when creating a new on-demand deployment for a regex branch pipeline.

Resolved in this release:

- Resources with a single parameter change are displayed correctly in impact analysis reports.
- Optional deployment plan parameters are no longer sent as empty strings to Bolt.
- Commit authors are shown when listing commits for Bitbucket Cloud repos during the creation of a new on-demand deployment or impact analysis report.
- Bitbucket Server URLs are rendered correctly in deployment approval emails.
- When logged in as a super user, clicking the **Workspaces** settings tab in the root console no longer changes the navigation panel options to those of a non-root workspace.

Version 3.6.1

Released 16 April 2020

Resolved in this release:

- Deployments using the temporary branch deployment policy now correctly perform a code deploy following a successful orchestrated deployment.

Version 3.6.0

Released 14 April 2020

New in this release:

- **Generate module impact analysis reports on demand.** You can now generate an impact analysis report for any module change by clicking **Manual actions** > **New Impact Analysis** on a module's details page.
- **Usability improvements.** Version 3.6.0 introduces several improvements to the design and usability of the web UI, including:
  - A redesigned **Users** page with new controls for adding users to your workspace or removing users from your workspace.
  - A redesigned **Groups** page with an updated workflow for adding and removing group members and user permissions.

Resolved in this release:

- Impact analysis reports no longer fail to generate if an impacted resource contains null Unicode characters.

Version 3.5.0

Released 2 April 2020

New in this release:

- **Usability improvements.** Version 3.5.0 introduces several improvements to the design and usability of the web UI, including:
  - The **SMTP Password** field now has controls to show or hide the password.

Resolved in this release:

- Changes to module resources are now correctly reflected in module impact analysis reports.
- Module deployments using the feature branch deployment policy now correctly create and name branches for GitHub and GitHub Enterprise users.
- The Continuous Delivery for PE server logs no longer include unnecessary `Unsupported PEM format` errors.

Version 3.4.1

Released 23 March 2020

Resolved in this release:

- Webhook-triggered jobs no longer fail for GitLab users.

Version 3.4.0

Released 18 March 2020

New in this release:

- **Use any node with a Puppet agent installed as job hardware.** You can now run your Continuous Delivery for PE jobs on any node with a Puppet agent installed. For instructions on configuring new job hardware, see

Configure job hardware running a Puppet agent. For instructions on migrating your existing job hardware servers, see Migrate job hardware.

> **Note:** To successfully configure a Puppet agent node, you must install the `puppetlabs-cd4pe_jobs` module and ensure the Continuous Delivery user role in PE can run the `cd4pe_jobs::run_cd4pe_job` task. See the documentation linked above for instructions.

- **Cancel an in-progress impact analysis task.** You can now cancel any scheduled or in-progress impact analysis task from the impact analysis details page.
- **Usability improvements.** Version 3.4.0 introduces several improvements to the design and usability of the web UI, including:
  - Clearer messaging in impact analysis reports and the removal of unhelpful "diff is too large" messages.
  - Deployments are no longer labeled **FAILED** when a deployment approval request is declined.
  - The **Control Repos** page now show 10 control repos per page, and the **Modules** page now shows 10 modules per page.
  - For deployments that require approval, the **Approve** and **Decline** buttons now vanish after an approval decision is provided.
  - Deployments that are awaiting approval now show a **PENDING APPROVAL** label instead of a **RUNNING** label in the **Events** timeline.
  - You can now enter either an IP address or hostname in the **Puppet Enterprise Console Address** field when adding new PE credentials.
- **Logging improvements.** Version 3.4.0 introduces several improvements to the Continuous Delivery for PE logs, including:
  - LDAP queries and replies are now included in the logs.

Resolved in this release:

- The GitHub `access_token` query parameter, which has been deprecated by GitHub, is no longer used by Continuous Delivery for PE in requests to the GitHub API.
- If the value of a port parameter in the `puppet_enterprise` class in the PE Infrastructure node group is set as a string, automatic integration of PE no longer fails.
- All events in a stage of a pipeline run no longer show the same timestamp.
- Version 3.4.0 resolves CVE-2020-7944. When you add a new resource or class with sensitive parameters, impact analysis reports redact the plain text values of the sensitive parameters.

Deprecated in this release:

- **Support for the Continuous Delivery agent on job hardware.** As part of our effort to simplify the Continuous Delivery for PE setup process and prioritize the tools PE users already have in place, support for the Continuous Delivery agent is deprecated in version 3.4.0, and will be removed in a future release. For more information, see Migrate job hardware.

Removed in this release:

- **Support for Puppet Enterprise version 2019.1.** PE 2019.1 has reached the end of its support lifecycle.
- **Hardware Agents.** As part of the deprecation of the Continuous Delivery agent, we've removed the **Hardware Agents** page from **Settings**.

Version 3.3.0

Released 19 February 2020

New in this release:

- **Include Bolt tasks in custom deployment policies.** You can now include Bolt tasks in the custom deployment policies you run in your Continuous Delivery for PE pipelines. If necessary, you can disable tasks by setting

`enable_pe_plans: false` in the `config` section of the `.cd4pe.yaml` file for the impacted control repo or module. For more on tasks, see Tasks and plans in PE.

CAUTION: Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.

- **Logging improvements.** Version 3.3.0 introduces several improvements to the Continuous Delivery for PE logs, including:
  - Information about control repo activities is now included in the logs.
  - Impact analysis information about file changes, module changes, and changed Hierakeys is now included in the logs.
  - To reduce unnecessary noise in the logs, log messages regarding dependency checking during pipeline runs are no longer included unless the logging level is increased to TRACE.

  For more about the Continuous Delivery for PE logs, see Troubleshooting.

Resolved in this release:

- Impact analysis reports that include Hiera data changes no longer include information on nodes impacted by the Hiera data change that are outside the selected environment.
- Module deployments using the feature branch policy no longer trigger the control repo pipeline associated with the feature branch policy.
- If a custom Docker image in the format `<IMAGE>:<VERSION>` is included in a job, webhooks for that job now fire correctly.

Version 3.2.1

Released 5 February 2020

Resolved in this release:

- **CVE-2020-7238.** This Netty vulnerability has been resolved.

Version 3.2.0

Released 4 February 2020

New in this release:

- **Feature branch deployment policy support for modules managed as code.** Deployments using the feature branch deployment policy can now be included in a module regex branch pipeline that is managed with a `.cd4pe.yaml` file.
- **Store custom deployment policies in `/site`.** Continuous Delivery for PE now looks for custom deployment policies in the `/site` directory of your control repo as well as in the `/module` and `/site-module` directories.
- **Usability improvements.** Version 3.2.0 introduces several improvements to the design and usability of the web UI, including:
  - When LDAP is enabled, the login screen asks for an LDAP username instead of an email address. This LDAP username maps to the **User attribute** setting from your LDAP configuration.
  - The YAML code validation tool shows an error message if your pipeline's YAML code includes an invalid regular expression.

Resolved in this release:

- The `deployment_policy_branch` parameter is now correctly applied when it is included in a `.cd4pe.yaml` file.
- Approval request emails are now delivered to members of the approval group.
- An appropriate message is shown in the web UI when a deployment approval request is declined.

Version 3.1.1

Released 28 January 2020

Resolved in this release:

- **CVE-2019-16869.** This Netty vulnerability has been resolved.

Version 3.1.0

Released 22 January 2020

New in this release:

- **Delete users.** Super users and the root user can now permanently delete user accounts from your Continuous Delivery for PE installation. Perform this action with caution: deleting a user also deletes all artifacts created by that user in Continuous Delivery for PE, including workspaces, jobs, integrations, pipelines, control repos, and module repos. For more information, see Delete a user.
- **Impact analysis includes Hiera data referenced in root-level `hiera.yaml` files.** Impact analysis reports now include changes to the Hiera data housed in locations referenced in the `hiera.yaml` file located at the root level of your control repo or module repo. If your control repo or module repo does not include a `hiera.yaml` file at the root level, Continuous Delivery for PE will fall back to analyzing Hiera changes in the `/data` and `/hieradata` directories.
- **Usability improvements.** Version 3.1.0 introduces several improvements to the design and usability of the web UI, including:
  - Better handling of long pipeline names.
  - Clearer messaging when creating a regex branch pipeline.
  - Validation of the selected Docker image name when a new Docker-based job is created.
  - An improved experience and clearer error message if a deployment fails because the target environment node group contains no nodes.

Resolved in this release:

- Impact analysis tasks can now be included in a module pipeline that is managed with a `.cd4pe.yaml` file.
- If a code manager task fails during a deployment attempt, the deployment details page now shows a **FAILED** status for that event instead of a **DONE** status.
- Newly created Docker-based jobs now use the correct default Docker image name.
- A duplicate description field is no longer present when you configure a manual deployment for a module.
- The associated control repo is automatically selected when you create an impact analysis stage in a module's pipeline.
- A deployment to a protected environment no longer shows a **PENDING** status after the deployment is approved.
- A Bolt error no longer occurs if a deployment using the temporary branch deployment policy is cancelled prior to the approval step.
- When Continuous Delivery for PE fails to correctly parse a `.cd4pe.yaml` file, it logs the parsing error in the application logs and displays it in the web UI.
- The status of Puppet runs is now correctly displayed on each deployment's details page.

Security notice:

- **CVE-2019-16869 is detectable in version 3.1.0.** A security scanner may detect a Netty vulnerability with a 5.0 CVSS score in Continuous Delivery for PE. However, Continuous Delivery for PE does not exercise the vulnerable code path and so is not vulnerable.

Deprecated in this release:

- **Support for MySQL and DynamoDB external databases.** As part of our effort to streamline the installation process and ensure Continuous Delivery for PE meets performance standards, support for MySQL and Amazon DynamoDB external databases is deprecated in version 3.1.0, and will be removed in a future release. Before support ends, we'll provide information about how to migrate your external database to a supported option.

Version 3.0.2

Released 19 December 2019

Resolved in this release:

- Deployments failed for any module regex branch deploying to a PE instance using prefixed environments where the selected prefix was "No prefix."

> **Note:** If you created a deployment of this type while using Continuous Delivery for PE version 3.0.0 or 3.0.1, you must delete and recreate the deployment for it to work properly.

- In control repo regex branch pipelines that were converted to management with code, deployments using the feature branch deployment policy failed validation.
- Module deployments could not be canceled.
- Control repo and module regex branch pipelines that are managed with code did not trigger correctly.
- Environment prefixes were not added to target environment names in deployments using the feature branch deployment policy from control repos. As a result, these deployments were not completed correctly.
- When a root or super user updated the Docker image used as global shared job hardware, the updated image was not used for jobs running on the shared job hardware.

Version 3.0.1

Released 16 December 2019

Resolved in this release:

- If you attempted to manage a pipeline as code that included a deployment using the feature branch policy, a `Parameter specified as non-null is null` error occurred and the pipeline did not successfully transition to management with code.
- Continuous Delivery for PE did not correctly default to looking for custom deployment policy files on the **Production** branch if a branch had not been set explicitly.

Version 3.0.0

Released 11 December 2019

New in this release:

- **Construct and manage your pipelines as code.** You now have the option to use a `.cd4pe.yaml` file housed in your control repo or module repo to construct, update, and manage your pipelines. Managing pipelines with code creates a version-controlled record of pipeline changes over time. For more information, see Constructing pipelines from code.
- **View Hiera changes in impact analysis reports.** When you update a YAML file in your Hiera data directory, impact analysis reports will now report what systems will be impacted and how their desired state will change. For the first version of this feature, Continuous Delivery for PE analyzes changes in `/data` and `/hieradata` directories in your control repo or module.

> **Important:** Hiera changes in impact analysis reports are only supported on PE 2019.2.0 and newer versions.

- **Usability improvements.** Version 3.0.0 introduces several improvements to the design and usability of the web UI, including:
  - A redesigned deployment details view featuring a new sequential list of the events that make up a deployment, with details about each event.
  - An updated pipelines design with clearer controls and a refreshed color palette.
  - A new **Manual actions** selector used for initiating on-demand impact analysis reports, deployments, or pipeline runs.
- **Improved deployment approval messaging.** The message sent to designated deployment approvers now contains more information about the proposed deployment, including the URL of the module or control repo, the

name of the user who initiated the deployment, the name of the pipeline, and a list of the commits included in the deployment.

- **Fewer stacktrace exceptions included in log files.** We've reduced the number of stacktrace exceptions that resulted from checking for dependencies and approvals. You'll no longer see long stacktrace errors for the following:

```
com.puppet.pipelines.cdpe.cdpeTaskUtils.CDPETaskInterruptedException:
 Dependency check attempt maxtime exceeded.
com.puppet.pipelines.cdpe.cdpeTaskUtils.CDPETaskInterruptedException:
 Approval check attempts maxiumum exceeded. Thread should yeild and try
 again.
```

Special beta feature in this release:

- **Custom deployment policies.** We've learned from our users that the deployment policies built into Continuous Delivery for PE don't always align with the deployment work you need to do. In response, we're introducing the ability to compose your own set of steps for deploying Puppet code. For more information, see Creating custom deployment policies.

> **CAUTION:** Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.

Resolved in this release:

- The name of the default Docker container is now consistently shown in the **Docker Image Name** field on the job creation page if no other Docker image is defined.

Security notices:

- **CVE-2019-16869 is detectable in version 3.0.0.** A security scanner may detect a Netty vulnerability with a 7.5 CVSS score in Continuous Delivery for PE. However, Continuous Delivery for PE does not exercise the vulnerable code path and so is not vulnerable.
- **Sonatype-2019-0115 is detectable in version 3.0.0.** This vulnerability is detected by the Sonatype Nexus scanner. However, Continuous Delivery for PE does not use the library that triggers the vulnerability and so is not vulnerable.

Removed in this release:

- **Incremental branch and blue-green branch deployment policies.** We've removed the incremental branch and blue-green branch deployment policies. If your pipeline included a deployment using one of these policies, the deployment has been removed from the pipeline. These policies were deprecated in Continuous Delivery for PE version 2.7.0.
- **Module deployment reports.** We've removed this feature from version 3.x.

Continuous Delivery for PE known issues

These are the known issues for the Continuous Delivery for PE 3.x release series.

Impact analysis tasks fail when using Puppet Enterprise versions 2021.2 or 2019.8.7

Impact analysis fails with a `R10K::Module::Forge cannot handle option 'default_branch_override'` error. If you're using these versions of PE, you must update the `pe-r10k` package by following the instructions in this Puppet Support article to continue to use impact analysis. After you update the package, you can update to future versions of PE using the installer as normal.

The `puppet_agent` module cannot be used to upgrade Puppet agents on job hardware

Due to the fact that Continuous Delivery for PE nodes are classified as PE infrastructure nodes, the `puppet_agent` module cannot be used to successfully upgrade Puppet agents running on job hardware. To upgrade the Puppet agent on a job hardware node, use the agent upgrade script.

A PE instance cannot be integrated if `dns_alt_names` is not set on the master certificate

If the Puppet master certificate for your PE instance does not have `dns_alt_names` configured, attempting to integrate the instance with Continuous Delivery for PE fails with a `We could not successfully validate the provided credentials against the Code Manager Service` error. The master certificate must be regenerated before PE is integrated with Continuous Delivery for PE. For instructions, see Regenerate master certificates in the PE documentation.

Impact analysis for Hiera data is unavailable when using PE 2019.7

If you are using PE version 2019.7 with Continuous Delivery for PE version 3.7.1 or earlier, impact analysis of Hiera data changes is unavailable. To resolve this issue, upgrade your Continuous Delivery for PE installation to version 3.8.0 or later.

Impact analysis report inaccuracies when using PE 2019.2 and earlier versions

Due to an r10k issue, impact analysis reports generated using PE 2019.2 and earlier versions might include nonexistent module changes and calculations of the impact of those nonexistent changes. The underlying issue was resolved in r10k version 3.4.0 and included in PE 2019.3 and all newer versions.

Unable to add Amazon S3 credentials when disk storage is configured

If you install Continuous Delivery for PE from the PE console (which automatically sets up disk storage), you might see an error if you attempt to add Amazon S3 credentials in the web UI. To work around this issue, sign into the root console and add your Amazon S3 credentials on the **Storage** tab of the **Settings** page.

Jobs fail when using chained SSL certificates on Windows

If you are using Continuous Delivery for PE with SSL configured to use chained certificates, attempts to run jobs on Puppet agent-based Windows job hardware will fail.

Rerun job control is unresponsive after two hours for Bitbucket Cloud users

This known issue applies only to Bitbucket Cloud users. When a pipeline run for a control repo or module was completed more than two hours ago, clicking the **Rerun Job** button results in an `Authentication failed` error.

Purging unmanaged firewall rules with the `puppetlabs-firewall` module deletes required firewall settings

If your Continuous Delivery for PE node uses the `puppetlabs-firewall` module to manage its firewall settings, and if a `resources { 'firewall': purge => true }` metaparameter is set on the node or at a higher level, Puppet will remove the unmanaged Docker firewall rules Continuous Delivery for PE requires to run successfully. To work around this issue, disable unmanaged firewall rule purging for your Continuous Delivery for PE node by changing the metaparameter to `resources { 'firewall': purge => false }`.

Deployments for module regex branches are not supported when managing pipelines as code in versions 3.0 and 3.1

In Continuous Delivery for PE versions prior to 3.2.0, deployments using the feature branch deployment policy cannot be included in a module regex branch pipeline that is managed with a `.cd4pe.yaml` file. To work around this issue, upgrade to version 3.2.0 or newer, or click **Manage pipelines** and select **Manage in the web UI**, then delete and recreate all deployments in the pipeline.

Module impact analysis tasks cannot be added to a pipeline after upgrading to version 3.0.0

If you added the credentials for the PE instance associated with a module pipeline's deployment tasks to Continuous Delivery for PE before you upgraded to version 3.0.0, you are unable to add impact analysis tasks to the pipeline. To work around this issue, delete and re-add the PE instance's credentials, giving the PE instance the same friendly name it had previously.

Custom deployment policies aren't initially shown for new control repos

When your first action in a newly created control repo is to add a deployment to a pipeline, any custom deployment policies stored in the control repo aren't shown as deployment policy options. To work around this issue, click **Built-in deployment policies**, then **Custom deployment policies** to refresh the list of available policies.

Regex branch module deployments fail if the `:control_branch` pattern is used for multiple modules

Deploying a module from a regex branch pipeline fails if more than one module in your Puppetfile uses the `:branch => :control_branch` pattern. To work around this issue, make sure that the `default_branch` parameter is set in the Puppetfile for every Git-sourced module that uses the `:branch => :control_branch` pattern.

Docker configuration changes to jobs are not immediately available

When you update the Docker configuration for a job, several minutes elapse before your changes take effect. To work around this issue, wait at least five minutes after making a Docker configuration change before attempting to run the job.

Users removed from all workspaces cannot add new workspaces

If you delete or are removed from all workspaces of which you are a member, you are directed to the **Add New Workspace** screen. If you log out or navigate away from this screen without creating a new workspace, you are unable to access any workspaces or get back to the **Add New Workspace** screen until invited to an existing workspace by another user. To work around this issue, create a new workspace when prompted, or request an invitation to an existing workspace.

Getting started with Continuous Delivery for PE

Greetings! Welcome to Continuous Delivery for PE. If you're trying out the software for the first time, this getting started guide is for you. As a new user, you'll need to perform some initial workspace setup tasks, and then we'll show you how to begin using core features of Continuous Delivery for PE.

You're just a few steps away from a more streamlined, powerful, and flexible Puppet code delivery process. Ready to get started?

Step 1: Install Continuous Delivery for PE

First, use the main documentation to install Continuous Delivery for PE.

Follow the instructions in Install Continuous Delivery for PE.

Step 2: Create your user account and set up a workspace

Think of a workspace like a neighborhood within the Continuous Delivery for PE city. Your workspace is where you store and access resources such as control repos, pipelines, and jobs. When you're ready to collaborate, you can invite the members of your team to join your workspace.

1. If you haven't already done so, navigate to the Continuous Delivery for PE web UI address you received at the end of the installation process in Step 1.
2. Create your user account.
   a) On the login page, click **Create an account**.
   b) Fill in the registration form and create a username and password.
   c) Click **Sign Up**.

   > **Note:** For almost all tasks you'll perform in Continuous Delivery for PE, you'll use your individual user account, not the root account. The root account is only used for special administrative tasks such as installation, designating super users, and deleting users.
3. Set up a workspace.
   a) On the **Choose a workspace** screen, click **+ Add new workspace**.
   b) Enter a name for your workspace and click **Create workspace**.

Step 3: Set up integrations

Next, it's time to set up integrations with your PE instance and the source control system where you keep your Puppet code.

We're sending you to our integration docs to complete these tasks.

1. Follow our Integrate with Puppet Enterprise instructions.
2. Follow the Configure impact analysis instructions.
3. Select your source control system from the list below and follow the integration instructions:
   - Azure DevOps Services
   - Bitbucket Cloud
   - Bitbucket Server
   - GitHub
   - GitHub Enterprise
   - GitLab
4. **Optional.** To make sure you're always running the latest available version of the software, install the `puppetlabs-cd4pe` module to automate upgrades of Continuous Delivery for PE.

Step 4: Configure job hardware

You'll now configure a node running a Puppet agent as a job hardware server, where your code will be tested before deployments.

1. Install a Puppet agent on the node you plan to use as job hardware. See Installing agents in the PE documentation for details.
2. Make sure your Continuous Delivery user role in PE includes the permission to run the `cd4pe_jobs::run_cd4pe_job` task.

3. Install the `puppetlabs-cd4pe_jobs` module, which is required to run Continuous Delivery for PE jobs on your node:
   a) Add the `puppetlabs-cd4pe_jobs` module to the following:
      - The Puppetfile for the production environment on the PE master that manages the agent node you've selected as job hardware
      - The Puppetfile on the `master` branch of the control repo you added to Continuous Delivery for PE in step 4

      A sample Puppetfile entry:

      ```
      mod 'puppetlabs-cd4pe_jobs', '1.5.0'
      ```

   b) Deploy the updated code to the production environment:

      ```
      puppet code deploy production --wait
      ```

4. We want this job hardware server to be able to use the pre-built jobs included in Continuous Delivery for PE. Since these jobs are Docker-based, you must install and configure Docker on the node. See the Installing Docker instructions for details.
5. Finally, tell Continuous Delivery for PE that this node is ready to be used as job hardware for Docker-based jobs by assigning it to a hardware capability. Capabilities organize your job hardware servers and ensure that jobs run on hardware with the right characteristics. Continuous Delivery for PE automatically creates a **Docker** hardware capability for you.
   a) In the Continuous Delivery for PE web UI, click **Hardware**.
   b) Locate the **Docker** capability and click **+ Edit**.
   c) Select the PE instance that manages the node you've selected as job hardware. Then, select your job hardware node.

      The selected node is added to the **Hardware with this capability** list on the right.
   d) Click **Save**.

Your job hardware node is now configured and ready for use. We'll see it in action during step 8, when we run our pipeline for the first time.

Step 5: Add a control repo

A control repo in Continuous Delivery for PE tracks the changes made on the active development branch of your source control system. When adding a control repo to Continuous Delivery for PE, it's important to connect the master Git branch.

When you set up your new control repo, Continuous Delivery for PE adds a webhook to the associated repository in your source control system. The webhook reports new commit activity on the repository to Continuous Delivery for PE, enabling you to track code changes and take action.

1. In the Continuous Delivery for PE web UI, click **Control repos**, then click **Add control repo**.
2. Follow the prompts to select your source control, organization, and your chosen repository.
3. The master branch of your repository is automatically selected for you. If your control repo does not currently contain a master branch, follow the prompts to let Continuous Delivery for PE create one for you, and select the branch that's under active development to create the master branch from.

> **Important:** When working with Continuous Delivery for PE, commit only to the master branch and to any feature branches (which are eventually merged back into the master branch). Do not push code changes to any of your other Git branches, as doing so can create conflicts with Continuous Delivery for PE workflows.

4. Optional: Edit the name of your new control repo.

> **Tip:** The control repo name must contain only alphanumeric characters, dashes, and underscores.

5. Click **Add**. The control repo is now shown in the master list on the **Control repos** page.

Step 6: Set up a pipeline

Pipelines in Continuous Delivery for PE are made up of stages and tasks. Tasks include jobs to test code, deployments, and impact analysis; stages group tasks into a series of sequential phases.

1. In the Continuous Delivery for PE web UI, click **Control repos**. Click the name of the control repo you added in Step 5.
2. On the right side of the web UI, you'll see the space where we'll create your pipeline. Click **+ Add default pipeline**.
3. Your default pipeline is automatically created for your master branch. This pipeline contains three stages:
   - The **Code validation stage** includes two jobs (tests for Puppet code).
   - The **Impact analysis stage** includes an impact analysis task with a pull request gate (more on this later).
   - The **Deployment stage**, where we'll add deployment instructions in step 8.

Step 7: Set up an environment node group

To give us a place to demonstrate how Continuous Delivery for PE deploys new code to your Puppet Enterprise-managed nodes, we'll next set up a small environment node group to use for deployment testing.

1. In your Git repository, create a new branch called **cd4pe_testing**. This will represent the environment node group.
2. On your master, run `puppet code deploy cd4pe_testing`.
3. In the PE console, click **Classification**.
4. Click **Add group...** and create a new node group with the following specifications:
   - Parent name: All Environments
   - Group name: CD4PE test group
   - Environment: cd4pe_testing
   - Environment group: yes
   - Description: Node group used for Continuous Delivery for PE testing

   Click **Add**.
5. In the list of node groups, click **CD4PE test group**. In the **Rules** tab, pin two or three test nodes to the node group. Make sure these nodes are not assigned to any other node groups in your PE installation.

Step 8: Deploy changes to your nodes

Once you've added a deployment to your pipeline, you can automatically move code changes to your nodes following the deployment conditions you've set.

1. In the Continuous Delivery for PE web UI, in your control repo's pipeline, click **Add a deployment**.
2. Select your PE instance, then select the **CDPE test group** node group we created in Step 7.
3. Select the **Direct deployment policy**. Don't worry about setting special conditions for this deployment.
4. Click **Add deployment to stage**. On the confirmation screen, click **Done**.
5. Now we need a change to pass through to production. In your Git repository, on the master branch, make a code change such as updating a package version. Commit the change and then return to the Continuous Delivery for PE web UI to watch your pipeline in action.

When triggered by your commit, the pipeline automatically runs tests on your code, skips over the impact analysis stage that we haven't yet set up, and stops, as the pipeline is set up not to autopromote into the Deployment stage.

Click **Promote** to continue the pipeline run and launch the deployment. The results of the pipeline run are logged in the **Events** area on the left of the screen.

Step 9: Create an impact analysis report

An impact analysis report shows the potential impact that new Puppet code will have on the nodes and resources you're managing with PE. You can add impact analysis tasks to your pipeline, and you can generate impact analysis reports on demand, as we'll do in this step.

1. First, create a change for the report to analyze. We'll generate an impact analysis report to review how this change impacts the nodes in your CDPE test group. In your Git repository, create a feature branch from your master branch
2. On the feature branch, make a code change, such as updating a package version.
3. Commit the change on the feature branch and then return to the Continuous Delivery for PE web UI.
4. In the Continuous Delivery for PE web UI, click **Manual actions** and select **New impact analysis**.
5. In the **New impact analysis** window, select your feature branch, then select the commit you just made. Select your PE instance and the CDPE test group node group.
6. Click **Analyze** to generate the report.

   Continuous Delivery for PE will now generate a new catalog containing your commit, and will compare this new catalog to the current catalog of the nodes in the CDPE test group.
7. Click **View impact analysis**. The report shows how the change on your feature branch would impact your nodes and resources if it was merged to the master branch.

Congratulations! You've reached the end of this introductory guide. You're now familiar with some of the core features of Continuous Delivery for PE, and have a basic understanding of how the software helps you deploy Puppet code and preview the impact of changes.

Installing

In order for your organization to begin using Continuous Delivery for Puppet Enterprise, you must first complete the initial installation and setup process.

To upgrade to the Continuous Delivery for PE 3.x series from a version in the 2.x series, see Upgrading to 3.x.

- Getting support on page 40

The Support team at Puppet provides support for all features and capabilities included in Continuous Delivery for PE. However, the Puppet Support team only supports the approved installation methods listed on this page, and cannot assist you with Docker configuration or container runtime issues unrelated to Continuous Delivery for PE.

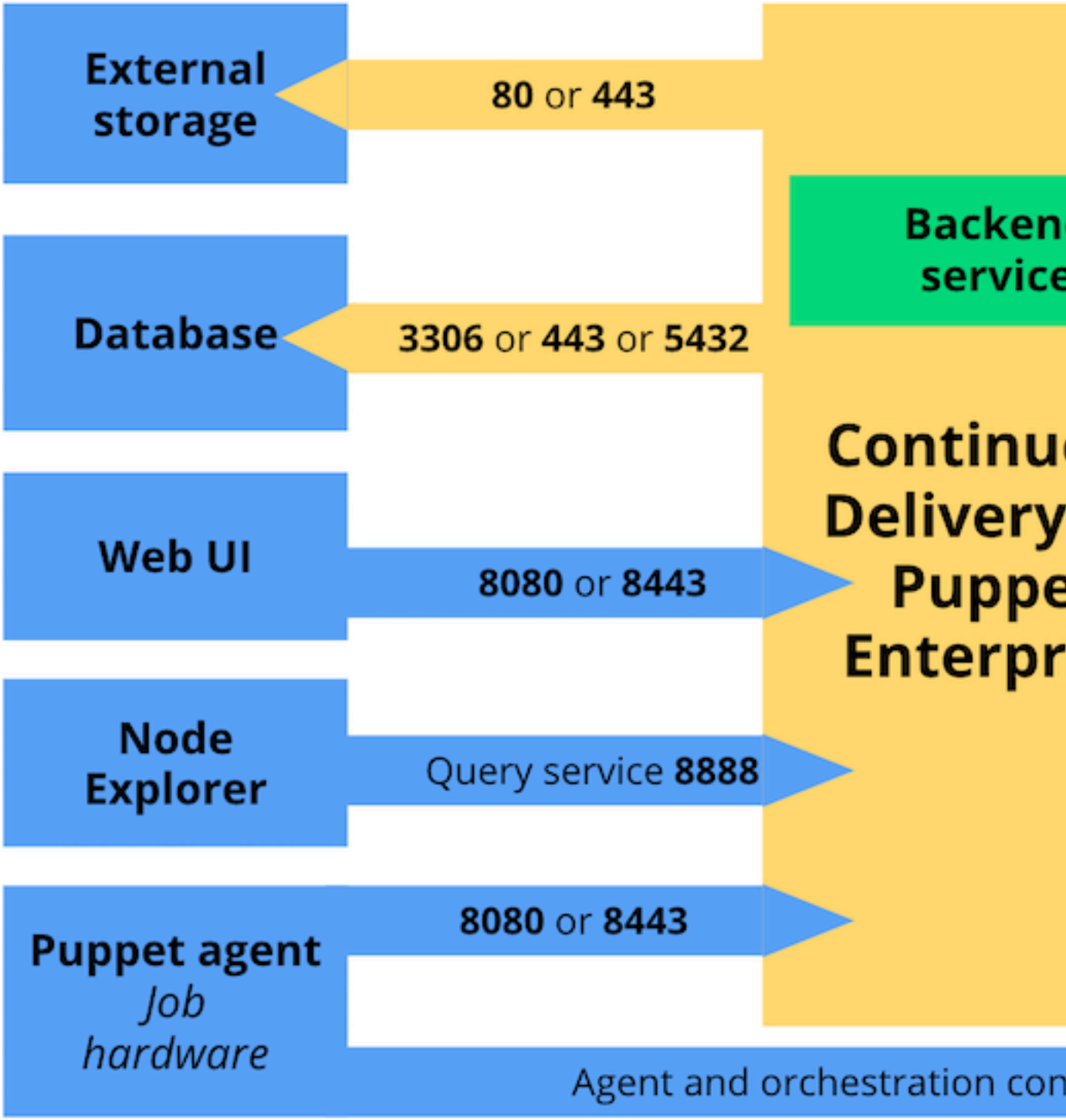
- Alternative installation methods on page 40

This section includes alternative methods for installing Continuous Delivery for Puppet Enterprise (PE).

Continuous Delivery for PE architecture

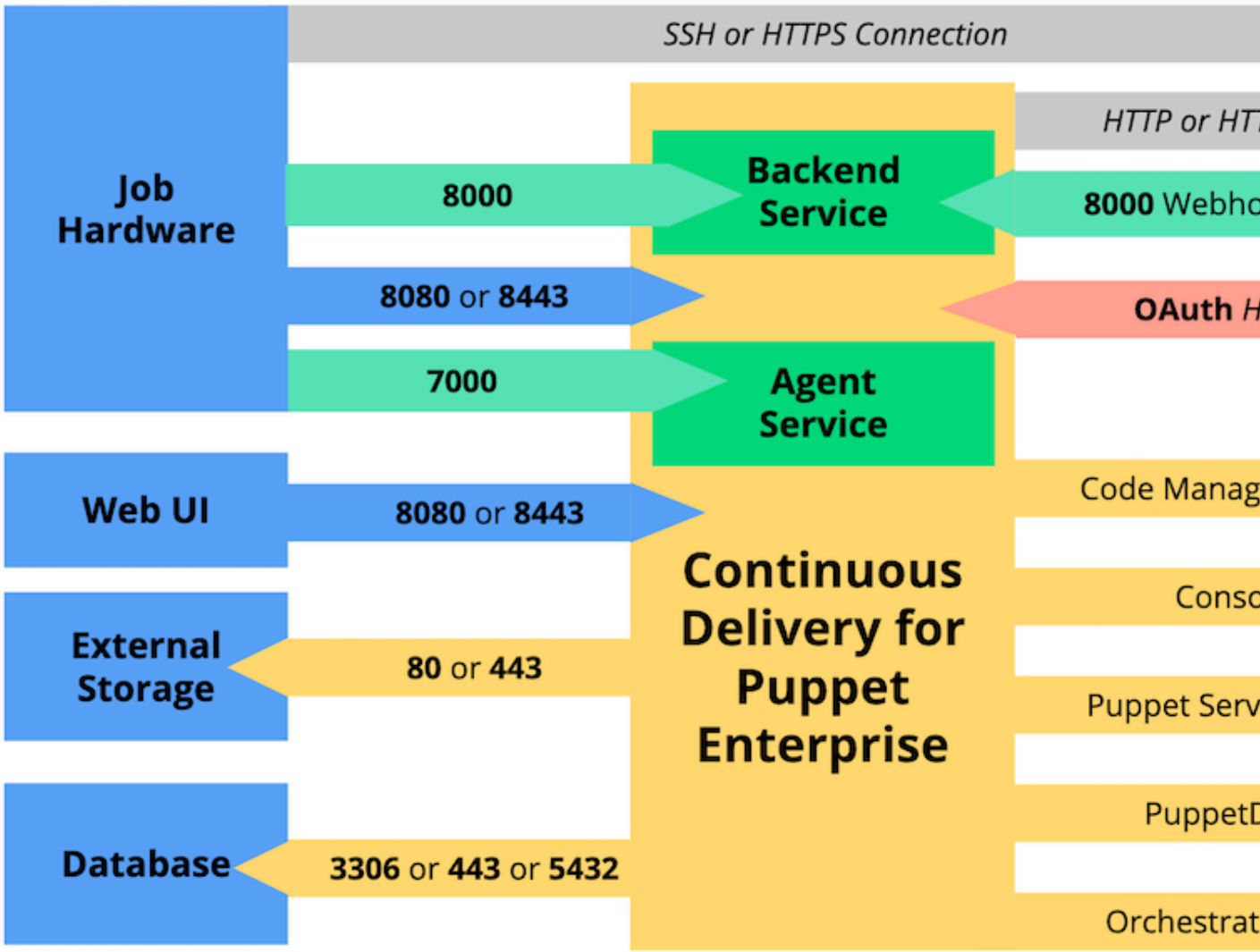
Continuous Delivery for Puppet Enterprise (PE) communicates with your PE installation, your source control system, and the servers you've designated as job hardware, as well as with the various components of the software.

The following diagram shows the architecture and port requirements of a Continuous Delivery for PE installation of version 3.4.0 or newer, using Puppet agent for job hardware.



The second diagram shows the architecture and port requirements of a typical Continuous Delivery for PE installation of version 3.3.0 or older, using the Continuous Delivery agent for job hardware.

Important: # The Continuous Delivery agent is deprecated as of Continuous Delivery for PE version 3.4.0, and will be removed in a future release.



Important: Continuous Delivery for PE uses TCP (Transmission Control Protocol) connections.

Job hardware and web UI configuration	Default port number
HTTP	8080
HTTPS	8443

External storage configuration	Default port number
Artifactory using HTTP	80
Artifactory using HTTPS	443
Amazon S3	443

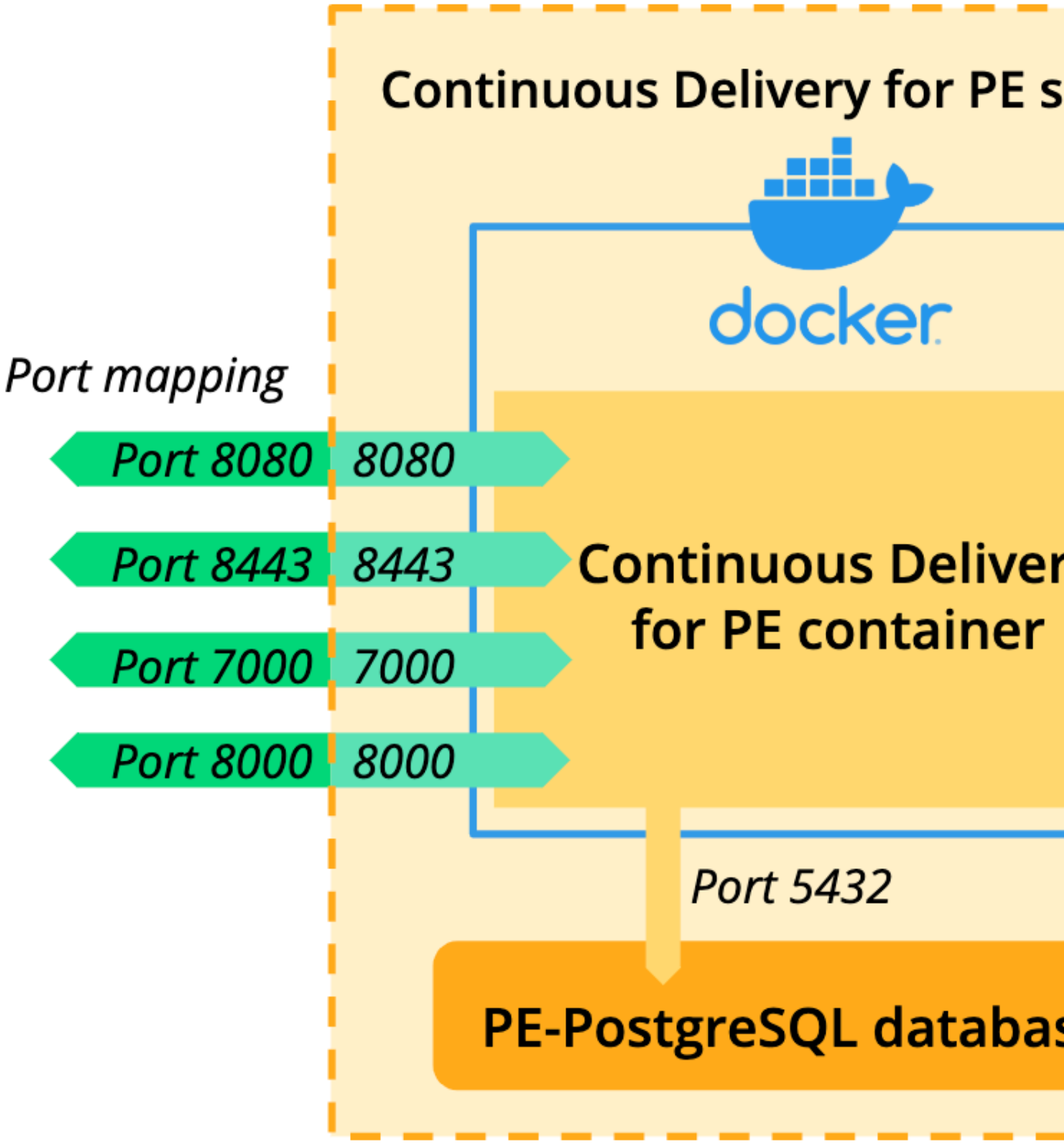
Continuous Delivery for PE uses a database to persist information. The `puppetlabs-cd4pe` module creates a new installation of PE-PostgreSQL on the node where you install Continuous Delivery for PE.

DEPRECATED: Support for external databases is deprecated, and will be removed in a future release.

Database configuration	Default port number
PE-PostgreSQL (local)	5432
# MySQL (external)	3306
# Amazon DynamoDB (external)	443

Continuous Delivery for PE Docker container configuration

Continuous Delivery for PE is run as a container in Docker. When you install the software, a PE-PostgreSQL database is created for you.



Use these environment variables to customize the Continuous Delivery for PE container:

Environment variable	Explanation
`ANALYTICS`	**Optional.** To opt out of analytics data collection, include `-e ANALYTICS=false`. To learn about what data we collect, see Analytics data collection.

Environment variable	Explanation
`DUMP_URI`	**Required.** How to address port 7000 of this container, which is the endpoint used by the Continuous Delivery for PE web UI to connect to the correct instance of the Continuous Delivery agent service. In a typical installation, this value is `dump://localhost:7000`.
`DB_ENDPOINT`	**Required if using an external database.** The `mysql://` or `ddb://` endpoint used to connect to your database. For example, `mysql://samplehost:3306/cdpe`.
`DB_USER` and `DB_PASS`	**Required if using an external database.** Credentials for your database user. **For MySQL:** Login credentials for your MySQL user. For security purposes, the database user you select should be able to connect to only this database. **For Amazon DynamoDB:** Access and secret keys for your DynamoDB user. **Important:** Make sure you generate credentials with full create, read, update, and delete permissions for DynamoDB resources. For security purposes, select a database user you select who can connect to only this database.
`DB_PREFIX`	**Optional.** When starting up, Continuous Delivery for PE creates tables in MySQL or DynamoDB. If you'd like the tables to share a prefix, such as `cdpe-`, enter it here. **Tip:** If you wish to simulate a fresh installation of a given version of Continuous Delivery for PE, entering a new database table prefix causes all the database tables to regenerate.

Environment variable	Explanation

`PFI_SECRET_KEY`	**Required.** A 16-byte secret key used for AES encryption of secrets (such as PE access tokens) supplied to Continuous Delivery for PE.
	
	If you're a *nix user, generate this key by running:
	
	`dd bs=1 if=/dev/urandom count=16 2>/dev/null | base64`
	
	If you're a Windows user, generate this key by running:
	
	`$randomBytes = New-Object Byte[](16)`
	`[Security.Cryptography.RNGCryptoServiceProvider]`
	`$encodedBytes =`
	`[System.Convert]::ToBase64String($randomBytes)`

The Continuous Delivery for PE container exposes these ports, which can be mapped to any ports of your choosing:

Port number	Port use
8080	Web UI (non-SSL access)
8443	Web UI (SSL access)
8000	Backend services
7000	Continuous Delivery agent service

System requirements

Refer to these system requirements for your Continuous Delivery for Puppet Enterprise (PE) installation.

Hardware requirements

Before installing Continuous Delivery for PE, ensure that your system meets these requirements.

Installation architecture	Memory	Storage	CPUs
Installation using disk object storage	8 GB	100 GB	4
Installation using external (Artifactory or Amazon S3) object storage	8 GB	50 GB	4

Supported operating systems

Continuous Delivery for PE can be installed on these operating systems. **The Continuous Delivery for PE host server must run the same operating system and version as your Puppet master.**

Operating system	Supported versions
Enterprise Linux • CentOS • Oracle Linux • Red Hat Enterprise Linux (RHEL) • Scientific Linux	7
Ubuntu (General Availability kernels)	16.04 (on PE versions prior to 2019.8), 18.04

External databases

DEPRECATED: Support for MySQL and Amazon DynamoDB external databases is deprecated, and will be removed in a future release. Information on how to migrate your external database to a supported option will be provided before support ends.

External database	Supported versions	Notes
MySQL	5.7	Your MySQL database must use the `latin1` character set and `latin1_swedish_ci` collation.
Amazon DynamoDB	n/a	Based on your usage, you might need to tune your tables' read/write capacity in order to reduce page load times.

Job hardware requirements

System requirements for your job hardware vary considerably based on the size of your Continuous Delivery for PE installation, the type of jobs you run, and how frequently you run them.

The size of the load placed on a job hardware server determines how robust that server's resources need to be. Determining that load involves a huge number of variables, from the number of jobs that run concurrently to the languages those jobs are written in. As a result, it's nearly impossible to provide one-size-fits-all system requirements for job hardware.

Instead, we've developed a sizing chart based on the estimated number of concurrent jobs a job hardware server is expected to regularly handle. While this chart represents our best estimates and understanding, it's provided only as a starting point. Testing and experience will help you fine-tune your job hardware and determine the optimum resource configuration for your installation's unique circumstances.

Estimated concurrent job load	Memory	Disk storage	CPUs
2 - 4 concurrent spec tests	4 GB	100 GB	2
4 - 8 concurrent spec tests	8 GB	100 GB	4
6 - 12 concurrent spec tests	8 GB	100 GB	6

When setting up your job hardware, keep these facts in mind:

- Disk storage requirements are minimal, and don't increase with added load. After a job run is complete, the job's log is passed to the object storage, and all data related to the job run is erased from the job hardware.
- You can run more jobs concurrently without increasing CPUs, but the jobs will run more slowly.

Job hardware requirements for Docker-based jobs

To run Docker-based jobs, your job hardware must have a modern version of Docker CE or Docker EE installed. The `puppetlabs/docker` module is our preferred way to install Docker and keep it up to date.

Job hardware used for Docker-based jobs also requires internet access. If you're working in an air-gapped environment, set up an internal Docker registry by following the Docker documentation.

Supported Puppet Enterprise versions

The following versions of Puppet Enterprise (PE) are supported for use with Continuous Delivery for PE.

PE version
2021.4
2021.3
2021.2
2021.1
2019.8.x (LTS)

For more on PE versions, see Puppet Enterprise support lifecycle.

Supported browsers

The following browsers are supported for use with the Continuous Delivery for PE web UI.

Browser	Supported versions
Google Chrome	Current version as of release
Mozilla Firefox	Current version as of release
Microsoft Edge	Current version as of release
Apple Safari	Current version as of release

Install Continuous Delivery for PE

Use version 1.1.0 or later of the `puppetlabs-cd4pe` module to install and configure Continuous Delivery for Puppet Enterprise (PE). This module installs Docker, configures the Continuous Delivery for PE Docker image and service for you, and creates a Docker volume for disk storage.

CAUTION: Code Manager webhooks are not compatible with Continuous Delivery for PE. If your organization currently uses Code Manager webhooks to deploy code, you must dismantle these webhooks before installing Continuous Delivery for PE.

Install Continuous Delivery for PE with the cd4pe module

Use the `puppetlabs-cd4pe` module version 1.1.0 or later to install Continuous Delivery for PE.

The `puppetlabs-cd4pe` module must be used with seven dependent modules, plus a module to manage job hardware. The modules and their required versions are as follows:

Module	Required version
`puppetlabs-cd4pe`	1.1.0 or later in the 1.x or 2.x series The **Nodes** page requires version 2.0.1 or later
`puppetlabs-stdlib`	4.19.0 or later

Module	Required version
puppetlabs-puppet_authorization	0.5.0 or later
puppetlabs-hocon	0.9.3 or later
puppetlabs-concat	2.1.0 or later
puppetlabs-docker	3.3.0 or later
puppetlabs-apt	4.4.1 or later
puppetlabs-translate	1.1.0 or later
puppetlabs-cd4pe_jobs	1.0.0 or later
Note: This module manages job hardware running a Puppet agent. Install this module with Continuous Delivery for PE version **3.4.0 or newer**.	
puppetlabs-pipelines	1.0.1
Note: This module manages job hardware running the deprecated Continuous Delivery agent. Install this module only if you are installing Continuous Delivery for PE version **3.3.0 or older**.	

1. Add the 10 modules listed above to the Puppetfile for each environment against which your compilers compile catalogs.

   A sample Puppetfile entry:

   ```
   mod 'puppetlabs-cd4pe', '2.0.2'
   # Requirements for cd4pe
   mod 'puppetlabs-stdlib', '8.1.0'
   mod 'puppetlabs-puppet_authorization', '0.5.1'
   mod 'puppetlabs-hocon', '1.1.0'
   mod 'puppetlabs-concat', '7.1.1'
   mod 'puppetlabs-docker', '4.1.2'
   mod 'puppetlabs-apt', '8.3.0'
   mod 'puppetlabs-translate', '2.2.0'
   mod 'puppetlabs-cd4pe_jobs', '1.5.0'
   # Required only if using Continuous Delivery agents for job hardware
   mod 'puppetlabs-pipelines', '1.0.1'
   ```

2. Deploy the updated code to the relevant environments by running `puppet code deploy <ENVIRONMENT>`.
3. In the PE console, click **Classification**. Click **Add group** and create a new node group with the following specifications.
   - **Parent name**: PE Infrastructure
   - **Group name**: Continuous Delivery for PE
   - **Environment**: production
   - **Environment group**: Do not select this option

4. Open the newly created Continuous Delivery for PE node group. Add the server you wish to use as your Continuous Delivery for PE host server to the node group by either creating a rule or pinning the node.

> **Important:** The Continuous Delivery for PE host server must run the same operating system and version as your Puppet master. Do not install Continuous Delivery for PE on the same server as your Puppet master.

5. Click Configuration. In the **Add new class** field, select the **cd4pe** class and click **Add class**.

> **Note:** If the cd4pe class isn't available, click **Refresh** to update the available class definitions.

6. Automate upgrades of Continuous Delivery for PE to the latest available version in the 3.x series by setting the `cd4pe_version` parameter to `3.x`. Add the parameter and commit your change.
After the classification change is applied on the next Puppet run, your Continuous Delivery for PE installation automatically upgrades itself whenever a new version is available.
7. Optional: Customize your Continuous Delivery for PE installation by setting any of the parameters listed in Advanced configuration options. If none of these parameters are set, your installation proceeds with the default settings.

> **Important:** In order to use the default installation, your Puppet certificate name must be a resolvable DNS hostname. If that is not the case, you must set the `resolvable_hostname` parameter to a resolvable address (hostname or IP address) where the Continuous Delivery for PE server is reachable. You can use the `${trusted[certname]}` fact to set this parameter.

8. Run Puppet on your Continuous Delivery for PE host server.

> **Note:** Once the Puppet agent run is complete, Docker downloads the Continuous Delivery for PE image, which can take a few minutes.

Continuous Delivery for PE is now installed.

See Configure Continuous Delivery for PE with a task to complete the configuration of the software and sign in.

Configure Continuous Delivery for PE with a task

Once you've completed the installation of Continuous Delivery for PE using the `cd4pe` module, run a task to configure the software.

> **Tip:** If you prefer to use classification, rather than a task, to configure your Continuous Delivery for PE installation, see Configure a Continuous Delivery for PE module installation using classification.

1. In the PE console, click **Classification**. Expand the **PE Infrastructure** node group and click **Continuous Delivery for PE**.
2. Click **Run** and select **Task**.
3. In the **Task** field, select `cd4pe::root_configuration`.

4. Enter parameters for the task, as follows:

Parameter	Value	Notes
root_email	The email address to associate with the root account.	Required.
root_password	The password to associate with the root account.	Required.
resolvable_hostname	The resolvable hostname where the Continuous Delivery for PE container can be reached. For example, if the container resides on a Docker host named `mydockerengine.myinc.com`, set resolvable_hostname to `http://mydockerengine.myinc.com`.	Required only if the agent certificate is not the machine's resolvable internet address.
agent_service_endpoint	The endpoint where the agent service can be reached, in the form `http://<resolvable_hostname>:<port>`.	Required if you set the agent_service_port parameter in the cd4pe class during installation.
backend_service_endpoint	The endpoint where the back end service can be reached, in the form `http://<resolvable_hostname>:<port>`.	Required if you set the backend_service_port parameter in the cd4pe class during installation.
web_ui_endpoint	The endpoint where the web UI can be reached, in the form `http://<resolvable_hostname>:<port>`.	Required if you set the web_ui_port parameter in the cd4pe class during installation.
storage_provider	Which object store provider to use. Must be one of: DISK, ARTIFACTORY or S3.	Defaults to DISK.
storage_bucket	The name of the bucket used for object storage.	Required if using Amazon S3 or Artifactory for object storage.
storage_endpoint	The URL of the storage provider.	Required if using Amazon S3 or Artifactory for object storage.
storage_prefix	For Amazon S3: the subdirectory of the bucket to use. For Artifactory: the top level of the Artifactory instance.	Optional.
s3_access_key	The AWS access key that has access to the bucket.	Required if using Amazon S3.
s3_secret_key	The AWS secret key that has access to the bucket.	Required if using Amazon S3.
artifactory_access_token	API token for your Artifactory instance.	Required if using Artifactory.

5. Click **Run job**.
6. When the job is complete, navigate to the URL printed on the task page. Click **Trial Mode** to start a free seven-day trial. Once this period is complete, you'll be prompted to generate and upload a license. See Generate a license for instructions on creating a free 30-day trial license.

Now that Continuous Delivery for PE is installed and configured, create your individual user account, then follow our Getting started with Continuous Delivery for PE guide starting at Step 3.

Advanced configuration options

Customize your Continuous Delivery for PE installation from the PE console by setting any of the following parameters on the `cd4pe` class. If none of these parameters are set, your installation proceeds with the default settings.

Parameters to configure the Docker image and version	
The following two parameters are concatenated by the `puppetlabs-cd4pe` module as follows: `image => "${cd4pe_image}:${cd4pe_version}",`	
cd4pe_image	Set this parameter if you use an internal Docker registry for mirroring containers. Use this parameter to set the image name; use cd4pe_version to set a tag.
cd4pe_version	Use this parameter to specify a particular version of the Continuous Delivery for PE Docker container. Specify `3.x` to use the 3.x series.

Parameters to configure the database	
# **DEPRECATED:** Support for MySQL and Amazon DynamoDB external databases is deprecated and will be removed in a future release. By default, the `puppetlabs-cd4pe` module (version 1.3.0 and newer) creates a new installation of PE-PostgreSQL on the node where you installed Continuous Delivery for PE. If you prefer to use Amazon DynamoDB or MySQL, set the parameters in this section. **CAUTION:** Changing any of these parameters post-install creates a new database and destroys all data kept in the previous database.	
manage_database	Set this parameter to `false` to use an external DynamoDB or MySQL server. Set this parameter to `true` to use Continuous Delivery for PE-managed PostgreSQL or MySQL.
db_provider	Enter `mysql` if you're using MySQL. Do not set this parameter if using DynamoDB.
db_host	Enter the address of the database. (Required for external MySQL and DynamoDB.)
db_name	Enter the name of the database. (Required for external MySQL and DynamoDB.)

Parameters to configure the database	
db_pass	Enter the password for the database. (Required for external MySQL and DynamoDB.)  **CAUTION:** To set your password successfully, you must set the root_password parameter to `Sensitive` in Hiera. For instructions, see Setting sensitive parameters in Hiera.
db_port	**Optional.** Enter the port the database listens on.
db_prefix	**Optional.** If you'd like your database tables to share a prefix, such as `cdpe-`, enter it here.

Parameters to configure the port mappings	
agent_service_port	Defaults to 7000.
backend_service_port	Defaults to 8000.
web_ui_port	Defaults to 8080.

Other optional parameters	
cd4pe_docker_extra_params	To pass any additional arguments to the Docker process running the Continuous Delivery for PE container, specify them as an array. For example: `["--add-host gitlab.puppetdebug.vlan:10.32.47.33","-v /etc/puppetlabs/cd4pe/config:/config","--env-file /etc/puppetlabs/cd4pe/env-extra","-e CD4PE_LDAP_GROUP_SEARCH_SIZE_LIMIT=250", ]`
analytics	To opt out of analytics data collection, set this parameter to `false`. To learn about what data we collect, see Analytics data collection.

Analytics data collection

Continuous Delivery for Puppet Enterprise (PE) automatically collects data about how you use the software. If you want to opt out of providing this data, you can do so when installing Continuous Delivery for PE.

Continuous Delivery for PE collects analytics data in order to better understand how our customers are using the software. For example, knowing how many control repos you manage helps us develop more realistic product testing. And learning which source control systems are the most and the least used helps us decide where to prioritize new functionality.

What data does Continuous Delivery for PE collect?

Continuous Delivery for PE collects analytics data when you use the software. No personally identifiable information is collected, and the data we collect is never used or shared outside Puppet.

The following data is collected when the software starts for the first time or restarts after an upgrade, and once per week thereafter:

- License UUID

- For each active user account:
  - Number of control repos
  - Number of control repo pipelines
  - Number of control repo pipelines with impact analysis enabled
  - Number of modules
  - The deployment policy selected for each pipeline deployment
  - Configured integrations (PE and source control systems)

The following data is collected while Continuous Delivery for PE is in use:

- Pageviews
- Progress through the initial installation and setup process
- Progress through the integration configuration process (PE and source control systems)

Continuous Delivery for PE **does not** collect:

- Any personally identifiable information about a user, such as name, email address, password, or company name
- User inputs such as usernames, control repo names, module names, job names, or job hardware information

Opt out of analytics data collection

To opt out of analytics data collection, use the PE console to set the `analytics` parameter on the `cd4pe` class.

1. Follow steps 1 through 5 of the installation instructions in Install Continuous Delivery for PE with the `cd4pe` module.
2. On the **Configuration** page of the Continuous Delivery for PE node group, add the **analytics** parameter to the **cd4pe** class. Set the parameter's value to `false`.
3. Run Puppet on your Continuous Delivery for PE node group to apply the change.
   You have opted out of data collection, and your new instance of Continuous Delivery for PE will not send analytics data to Puppet.

Puppet License Manager

Puppet License Manager is a centralized hub for self-service license management for Puppet products. Sign into Puppet License Manager and use it to create and manage license files for Continuous Delivery for Puppet Enterprise.

Create a personal Puppet account

To use Puppet License Manager, you must create a Puppet account.

1. Navigate to licenses.puppet.com.
2. Click **Sign Up** and fill out the registration form.
3. Click **Sign Up**.

Your account is created, and the main license management screen opens.

> **Important:** If you need to change your password, you can do so from the Puppet License Manager login screen.

Create a team Puppet account

If you wish to share access to product licenses with other members of your team or organization, create a team account.

> **Before you begin**
>
> Each member of the team must create a personal Puppet account.

1. Navigate to licenses.puppet.com.
2. Click **Sign Up** and fill out the registration form.
3. Click **Sign Up**.
4. Add members to the team account. In the Puppet License Manager web UI, click **Settings**  , then click **Add Account Member**.
5. Enter the email address of a team member and click **Add User**. Repeat this step for each member of your team.
6. When you're finished adding users, click **Cancel** to exit the **Add User** pane.

The **Account Members** screen now shows a list of the members of your team account. You can add or remove account members from this screen at any time.

Access a team Puppet account

After you've been added as a member of a team account, you can switch between your personal and team accounts without logging out of Puppet License Manager.

1. Navigate to licenses.puppet.com.
2. Sign into your personal Puppet account.
3. In the Puppet License Manager web UI, click your username  to open the list of accounts you have access to. You'll see your personal account and any team accounts you're a member of.
4. Select a personal or team account username from the list to switch to that account.

The username for the account you're viewing is always displayed in the navigation bar.

Generate and download a trial license

Puppet offers a 30-day trial license for Continuous Delivery for Puppet Enterprise. Generate a trial license with Puppet License Manager.

1. Sign into Puppet License Manager by entering your Puppet account credentials.

> **Tip:** To generate a license for a team account, sign in with your personal Puppet account. In the Puppet License Manager web UI, switch to the appropriate team account by clicking your username  to open the list of accounts you have access to.

2. Click **Get License**.
3. Select the product you want to try out and click **30-day Free Trial**.
4. Complete the license registration form.
5. Click **Start 30-day Trial**.

   Your license is created, and the main license management screen opens.
6. Click **Download** and save the license file in a secure place. You'll be asked to upload it when installing Continuous Delivery for PE.

Viewing active and expired licenses

View a list of your licenses, along with their creation and expiration dates and current status, by visiting the **Licenses** screen in Puppet License Manager.

Getting support

The Support team at Puppet provides support for all features and capabilities included in Continuous Delivery for PE. However, the Puppet Support team only supports the approved installation methods listed on this page, and cannot assist you with Docker configuration or container runtime issues unrelated to Continuous Delivery for PE.

Supported installation methods

The Support team can assist with the Continuous Delivery for PE installation process using the following supported methods:

- Installation from the PE console
- Installation with the `cd4pe` module
- Installation with Docker

> **Note:** The Support team can help you install Continuous Delivery for PE with Docker using Docker CE or Docker EE, but cannot help with issues not directly related to the software, such as configuring systemd services, persistent volumes, or Docker networking.

Although Continuous Delivery for PE is run as a Docker container, the Support team cannot help you deploy the application to all types of container runtime, and cannot assist with container runtime-related issues.

Unsupported installation methods

Unsupported installation methods include, but are not limited to:

- Kubernetes
- Pivotal Cloud Foundry
- Mesosphere

Application support

The Puppet Support team supports all Continuous Delivery for PE features and capabilities, regardless of where the application is running. However, if a feature is not working correctly due to your runtime environment, the Support team reserves the right to reject the support request.

Alternative installation methods

This section includes alternative methods for installing Continuous Delivery for Puppet Enterprise (PE).

- Install Continuous Delivery for PE in an offline environment on page 40

Use these instructions to install Continuous Delivery for PE in an air-gapped or offline environment where the Continuous Delivery for PE host node does not have direct access to the internet.

- Configure a Continuous Delivery for PE module installation using classification on page 43

Once you've completed the installation of Continuous Delivery for PE using the `cd4pe` module, you can use classification to configure the software.

Install Continuous Delivery for PE in an offline environment

Use these instructions to install Continuous Delivery for PE in an air-gapped or offline environment where the Continuous Delivery for PE host node does not have direct access to the internet.

1. Using a proxy server or a computer with internet access, download Docker CE.

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/
centos/docker-ce.repo
sudo yumdownloader --resolve docker-ce
```

2. Using a proxy server or a computer with internet access, download the Continuous Delivery for PE container image.

```
sudo docker pull puppet/continuous-delivery-for-puppet-enterprise:3.x
sudo docker image save puppet/continuous-delivery-for-puppet-enterprise -o
 cd4pelatest.tar
```

3. Transfer the Continuous Delivery for PE container image to the server you've designated as the Continuous Delivery for PE host.

```
scp cd4pelatest.tar <CD4PE_HOST_ADDRESS>
sudo docker image load -i /<directory>/cd4pelatest.tar
```

4. Using a proxy server or a computer with internet access, download the `puppetlabs-cd4pe` module, its dependent modules, and a module to manage job hardware.

   The modules and their required versions are as follows:

Module	Required version
`puppetlabs-cd4pe`	1.1.0 or later in the 1.x or 2.x series The **Nodes** page requires version 2.0.1 or later
`puppetlabs-stdlib`	4.19.0 or later
`puppetlabs-puppet_authorization`	0.5.0 or later
`puppetlabs-hocon`	0.9.3 or later
`puppetlabs-concat`	2.1.0 or later
`puppetlabs-docker`	3.3.0 or later
`puppetlabs-apt`	4.4.1 or later
`puppetlabs-translate`	1.1.0 or later
`puppetlabs-cd4pe_jobs`	1.0.0 or later
Note: This module manages job hardware running a Puppet agent. Install this module with Continuous Delivery for PE version **3.4.0 or newer**.	
`puppetlabs-pipelines`	1.0.1
Note: This module manages job hardware running the deprecated Continuous Delivery agent. Install this module only if you are installing Continuous Delivery for PE version **3.3.0 or older**.	

5. Add the nine modules listed above to the Puppetfile for each environment against which your compilers compile catalogs.

   A sample Puppetfile entry:

   ```
   mod 'puppetlabs-cd4pe', '2.0.2'
   # Requirements for cd4pe
   mod 'puppetlabs-stdlib', '8.1.0'
   mod 'puppetlabs-puppet_authorization', '0.5.1'
   mod 'puppetlabs-hocon', '1.1.0'
   mod 'puppetlabs-concat', '7.1.1'
   mod 'puppetlabs-docker', '4.1.2'
   mod 'puppetlabs-apt', '8.3.0'
   mod 'puppetlabs-translate', '2.2.0'
   mod 'puppetlabs-cd4pe_jobs', '1.5.0'
   # Required only if using Continuous Delivery agents for job hardware
   mod 'puppetlabs-pipelines', '1.0.1'
   ```

6. Deploy the updated code to the relevant environments by running `puppet code deploy <ENVIRONMENT>`.
7. Configure the Continuous Delivery for PE host node:
   a) Install the operating system and version used on your Puppet master.

   > **Important:** The Continuous Delivery for PE host server must run the same operating system and version as your Puppet master. Do not install Continuous Delivery for PE on the same server as your Puppet master.

   b) Install a Puppet agent. See Installing agents in the PE documentation for details.
   c) Install and start Docker CE . Use the `scp` command to move the Docker CE package you downloaded in step 1 to the Continuous Delivery for PE host node.

   ```
   sudo rpm -ivh *.rpm
   sudo systemctl start docker
   ```

8. In the PE console, click **Classification**. Click **Add group** and create a new node group with the following specifications.
   - **Parent name**: PE Infrastructure
   - **Group name**: Continuous Delivery for PE
   - **Environment**: production
   - **Environment group**: Do not select this option
9. Open the newly created Continuous Delivery for PE node group. Add your Continuous Delivery for PE host server to the node group by either creating a rule or pinning the node.
10. Click Configuration. In the **Add new class** field, select the **cd4pe** class and click **Add class**.

> **Note:** If the cd4pe class isn't available, click **Refresh** to update the available class definitions.

11. Automate upgrades of Continuous Delivery for PE to the latest available version in the 3.x series by setting the `cd4pe_version` parameter to `3.x`. Add the parameter and commit your change.
    After the classification change is applied on the next Puppet run, your Continuous Delivery for PE installation automatically upgrades itself whenever a new version is available.
12. Optional: Customize your Continuous Delivery for PE installation by setting any of the parameters listed in Advanced configuration options. If none of these parameters are set, your installation proceeds with the default settings.

> **Important:** In order to use the default installation, your Puppet certificate name must be a resolvable DNS hostname. If that is not the case, you must set the `resolvable_hostname` parameter to a resolvable address (hostname or IP address) where the Continuous Delivery for PE server is reachable. You can use the `${trusted[certname]}` fact to set this parameter.

13. Run Puppet on your Continuous Delivery for PE host server.

> **Note:** This Puppet run will return "unreachable repository" errors, which are expected at this point in this process.

14. Install the PE-PostgreSQL components needed to set up the local database.

```
yum --disablerepo=docker -d 0 -e 0 -y install pe-postgresql96-server
yum --disablerepo=docker -d 0 -e 0 -y install pe-postgresql96-contrib
```

15. Run Puppet on your Continuous Delivery for PE host server. This run should complete without any errors or warnings about skipped steps.
16. Finally, follow the instructions in Configure Continuous Delivery for PE with a task. When configuration is complete, you'll be directed to navigate to the Continuous Delivery for PE web UI.

The installation process is now complete, and you're ready to run Continuous Delivery for PE in an offline environment.

Configure a Continuous Delivery for PE module installation using classification

Once you've completed the installation of Continuous Delivery for PE using the `cd4pe` module, you can use classification to configure the software.

1. In the PE console, click **Classification**. Expand the **PE Infrastructure** node group and click **Continuous Delivery for PE**.
2. Click **Configuration**. In the **Add new class** field, select **cd4pe::root_config**.

3. Enter parameters for the class, as follows:

Parameter	Value	Notes
root_email	The email address to associate with the root account.	Required.
root_password	The password to associate with the root account. **CAUTION:** To set your password successfully, you must set the root_password parameter to `Sensitive` using Hiera. This parameter cannot be set successfully in the PE console. For instructions, see Setting sensitive parameters in Hiera.	Required. **Note:** If you need to update the root account password, first do so in the Continuous Delivery for PE web UI, and then update the password in the PE console using this parameter.
storage_provider	Which object store provider to use. Must be one of: DISK, ARTIFACTORY or S3.	Defaults to DISK.
storage_bucket	The name of the bucket used for object storage.	Required if using Amazon S3 or Artifactory for object storage.
storage_endpoint	The URL of the storage provider.	Required if using Amazon S3 or Artifactory for object storage.
storage_prefix	For Amazon S3: the subdirectory of the bucket to use. For Artifactory: the top level of the Artifactory instance.	Optional.
s3_access_key	The AWS access key that has access to the bucket.	Required if using Amazon S3.
s3_secret_key	The AWS secret key that has access to the bucket.	Required if using Amazon S3.
artifactory_access_token	API token for your Artifactory instance.	Required if using Artifactory.

4. Commit your changes and run Puppet on the node group.
5. When the Puppet run is complete, navigate to port 8080 of the Continuous Delivery for PE host server. Click **Trial mode** to start a free seven-day trial. Once this period is complete, you'll be prompted to generate and upload a license. See Generate a license for instructions on creating a free 30-day trial license.

Now that Continuous Delivery for PE is installed and configured, create your individual user account and then move on to these next steps:

- Integrate your source control system.
- Integrate with Puppet Enterprise.
- Configure impact analysis.
- Configure job hardware, which is used when testing your Puppet code.

- Follow our Getting started with Continuous Delivery for PE guide to learn about core workflows and capabilities.

Setting sensitive parameters in Hiera

When passing sensitive information, such as the root password for Continuous Delivery for PE, as parameters, set the value of these parameters as `Sensitive` in Hiera.

To set the value of parameters from `String` to `Sensitive` in Hiera, add the parameters to a `lookup_options` section in the `common.yaml` file. For example:

```
---
lookup_options:
  '^cd4pe::db_pass$':
    convert_to: 'Sensitive'
  '^cd4pe::root_config::root_password$':
    convert_to: 'Sensitive'
  '^cd4pe::root_config::s3_secret_key$':
    convert_to: 'Sensitive'
  '^cd4pe::root_config::artifactory_access_token$':
    convert_to: 'Sensitive'
```

Upgrading

New versions of Continuous Delivery for Puppet Enterprise (PE) are released regularly. Upgrading to the current version ensures you're always taking advantage of the latest features, fixes, and improvements.

Continuous Delivery for PE runs inside a Docker container, and new versions are released to Docker Hub. Upgrading to the latest version of Continuous Delivery for PE requires destroying the current Docker container and building a new container in its place that includes the latest version of the software. The `puppetlabs-cd4pe` module includes a `docker-cd4pe` service that manages this process for you; once the module is installed on your system, when you stop and restart the `docker-cd4pe` service, the container rebuild process is completed automatically.

Upgrading to the latest version of Continuous Delivery for PE

If you used the `puppetlabs-cd4pe` module to install Continuous Delivery for PE, stop and restart the `docker-cd4pe` service with either `systemctl` or Puppet.

With `systemctl`:

- Stop the service: `systemctl stop docker-cd4pe`
- Restart the service: `systemctl start docker-cd4pe`

With Puppet:

- Stop the service: `puppet resource service docker-cd4pe ensure=stopped`
- Restart the service: `puppet resource service docker-cd4pe ensure=running`

If you installed Continuous Delivery for PE from the PE console, install the `puppetlabs-cd4pe` module to automate upgrades. Then, stop and restart the `docker-cd4pe` service with either `systemctl` or Puppet.

With `systemctl`:

- Stop the service: `systemctl stop docker-cd4pe`
- Restart the service: `systemctl start docker-cd4pe`

With Puppet:

- Stop the service: `puppet resource service docker-cd4pe ensure=stopped`
- Restart the service: `puppet resource service docker-cd4pe ensure=running`

> **Note:** Make sure that the value of the **cd4pe_version** parameter in **Class:cd4pe** is set to `3.x` and not pinned to a specific version.

Automate upgrades of Continuous Delivery for PE

Install the `puppetlabs-cd4pe` module, then create and classify a node group to automate management of your Continuous Delivery for PE installation's version.

1. Add the module and its dependencies to your Puppetfiles.

   The `puppetlabs-cd4pe` module must be used with seven dependent modules, plus a module to manage job hardware. The modules and their required versions are as follows:

Module	Required version
`puppetlabs-cd4pe`	1.1.0 or later in the 1.x or 2.x series The **Nodes** page requires version 2.0.1 or later
`puppetlabs-stdlib`	4.19.0 or later
`puppetlabs-puppet_authorization`	0.5.0 or later
`puppetlabs-hocon`	0.9.3 or later
`puppetlabs-concat`	2.1.0 or later
`puppetlabs-docker`	3.3.0 or later
`puppetlabs-apt`	4.4.1 or later
`puppetlabs-translate`	1.1.0 or later
`puppetlabs-cd4pe_jobs` **Note:** This module manages job hardware running a Puppet agent. Install this module with Continuous Delivery for PE version **3.4.0 or newer**.	1.0.0 or later
`puppetlabs-pipelines` **Note:** This module manages job hardware running the deprecated Continuous Delivery agent. Install this module only if you are installing Continuous Delivery for PE version **3.3.0 or older**.	1.0.1

2. Add the 10 modules listed above to the Puppetfile for each environment against which your compilers compile catalogs.

   A sample Puppetfile entry:

   ```
   mod 'puppetlabs-cd4pe', '2.0.2'
   # Requirements for cd4pe
   mod 'puppetlabs-stdlib', '8.1.0'
   mod 'puppetlabs-puppet_authorization', '0.5.1'
   mod 'puppetlabs-hocon', '1.1.0'
   mod 'puppetlabs-concat', '7.1.1'
   mod 'puppetlabs-docker', '4.1.2'
   mod 'puppetlabs-apt', '8.3.0'
   mod 'puppetlabs-translate', '2.2.0'
   mod 'puppetlabs-cd4pe_jobs', '1.5.0'
   # Required only if using Continuous Delivery agents for job hardware
   mod 'puppetlabs-pipelines', '1.0.1'
   ```

3. Deploy the modules to the `production` environment by running `puppet code deploy production`.
4. In the PE console, click **Classification**. Click **Add group** and create a new node group with the following specifications.
   - **Parent name**: PE Infrastructure
   - **Group name**: Continuous Delivery for PE
   - **Environment**: production
   - **Environment group**: Do not select this option
5. Open the newly created Continuous Delivery for PE node group. Add your Continuous Delivery for PE host server to the node group by either creating a rule or pinning the node.
6. Click **Configuration**. In the **Add new class** field, select the **cd4pe** class and click **Add class**.

   > **Note:** If the **cd4pe** class isn't available, click **Refresh** to update the available class definitions.

7. To automate upgrades of Continuous Delivery for PE to the latest available version in the 3.x series, set the **cd4pe_version** parameter to `3.x`. Add the parameter and commit your change.

After the classification change is applied on the next Puppet run, your Continuous Delivery for PE installation automatically upgrades itself whenever a new version is available and you restart the `docker-cd4pe` service managed by the module.

Upgrade to the 3.x series

Upgrades from the Continuous Delivery for Puppet Enterprise (PE) 2.x series to the 3.x series are not automatic. Instead, you must upgrade your version of the software by updating your `puppetlabs-cd4pe` module classification.

If you installed Continuous Delivery for PE from the PE console or by using the `puppetlabs-cd4pe` module, follow these instructions to upgrade to the 3.x series.

1. In the PE console, click **Classification**.
2. Expand the **PE Infrastructure** group, and select the **Continuous Delivery for PE** node group.
3. Click **Configuration**.
4. Under **Class:cd4pe**, in the **Parameter** field, select **cd4pe_version**. In the **Value** field, enter `3.x`. Click **Add parameter** and commit your changes.
5. To apply the new version, run Puppet on the node group. At the top of the page, click the **Run** selector and choose **Puppet**. On the **Run Puppet** page that opens, make any necessary adjustments and click **Run Job**.

When the Puppet run is complete, your Continuous Delivery for PE instance will be running the 3.x series. You'll now automatically upgrade to new 3.x series versions as they are released whenever you stop and restart the `docker-cd4pe` service managed by the module.

Configuring and adding integrations

Configure your Continuous Delivery for Puppet Enterprise (PE) instance so that it communicates with your source control system, Puppet Enterprise, and the job hardware you use to run tests on your Puppet code.

Integrate with Puppet Enterprise

To set up an integration between your Puppet Enterprise (PE) instance and Continuous Delivery for PE, you must first set up a dedicated PE user with appropriate permissions, then add your PE instance's credentials to Continuous Delivery for PE.

Before you begin

- You must enable Code Manager on the PE instance before integrating with Continuous Delivery for PE. For instructions, see Configuring Code Manager.
- The Puppet master certificate must have `dns_alt_names` configured. To confirm whether your certificate is configured correctly, run:

```
openssl x509 -in $(puppet config print hostcert) -text |grep -A 1 "Subject
 Alternative Name"
```

If no output is returned, the master certificate must be regenerated before PE is integrated with Continuous Delivery for PE. For instructions, see Regenerate master certificates in the PE documentation.

> **Note: For PE instances with a replica configured for disaster recovery.** In the event of a partial failover, Continuous Delivery for PE is not available. Learn more at What happens during failovers in the PE documentation. To restore Continuous Delivery for PE functionality, you must promote the replica to primary server.

Create a Continuous Delivery user and user role in PE

Create a "Continuous Delivery" user and user role in PE. This allows you to view a centralized log of the activities Continuous Delivery for PE performs on your behalf. You'll also use this user account when generating the PE authentication token required by the setup process.

1. To begin, create a new user. In the PE console, click **Access control** > **Users**.
2. Enter a full name (such as `Continuous Delivery User`) and a login name (such as `cdpe_user`) and click **Add local user**.
3. Next, create a user role containing the permissions the Continuous Delivery User needs when operating Continuous Delivery for PE. In the PE console, click **Access control** > **User roles**.
4. Enter a name and (optional) description for new role, such as `CDPE User Role`, then click **Add role**.
5. Select the user role you've just created from the list on the **User roles** page.
6. Click **Permissions**. Assign the following permissions to the user role:

Type	Permission	Object
Job orchestrator	Start, stop, and view jobs	-
Node groups	Create, edit, and delete child groups	All
Node groups	View	All
Node groups	Edit configuration data	All
Node groups	Set environment	All
Nodes	View node data from PuppetDB	-
Puppet agent	Run Puppet on agent nodes	-
Puppet environment	Deploy code	All
Puppet Server	Compile catalogs for remote nodes	-
Tasks	Run tasks	At a minimum, `cd4pe_jobs::run_cd4pe_job`, which is required when running job hardware using a Puppet agent.

7. Once all the permissions have been added, click **Commit changes**.
8. Add your Continuous Delivery user to the user role. Click **Member users**. Select the name of the user you created earlier, and click **Add user**, then commit your change.
9. Your user is now set up and has been given the permissions needed to operate Continuous Delivery for PE. Before proceeding, create a password for the Continuous Delivery user.
   a) Return to the **Users** page.
   b) Find and click the full name of your newly created user, then click **Generate password reset**.
   c) Follow the link created and create a password for the user. You'll use this password when adding your PE credentials to Continuous Delivery for PE.

Add your Puppet Enterprise credentials

Establishing an integration with your Puppet Enterprise (PE) instance allows Continuous Delivery for PE to work with PE tools such as Code Manager and the orchestrator service to deploy Puppet code changes to your nodes.

If necessary, you can add multiple PE instances to your Continuous Delivery for PE installation.

> **Important:** Do not perform these steps while signed in as the root user. Sign into Continuous Delivery for PE with your individual user account before proceeding.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Puppet Enterprise**. Click **+ Add new credentials**.
3. In the **New Puppet Enterprise credentials** pane, enter a unique friendly name for your Puppet Enterprise installation.

   If you need to work with multiple PE installations within Continuous Delivery for PE, these friendly names help you to differentiate which installation's resources you're managing, so choose them carefully.
4. Enter the fully qualified domain name (FQDN) you use to access the PE console (such as `sample.pe.instance`). The FQDN must match the certname of your PE master or an alias included in the `dns_alt_names` entry in your `puppet.conf` file.
5. Select **Basic authorization** or **API token** and enter the required information:

   - For **Basic authorization**, enter the username and password for your "Continuous Delivery" user. Continuous Delivery for PE uses this information to generate an API token for you. The username and password are not saved. Optionally, change the token's lifetime by clicking **Change**.
   - For **API token**, generate a PE access token for your "Continuous Delivery" user using `puppet-access` or the RBAC v1 API, and paste it into the **API token** field.

     For instructions on generating an access token, see Token-based authentication.

     > **Tip:** To avoid unintended service interruptions, create an access token with a multi-year lifespan.
6. Click **Save changes**.

   Continuous Delivery for PE uses the information you provide to look up the endpoints for PuppetDB, Code Manager, orchestrator, and node classifier, and to access the master SSL certificate generated during PE installation. Once your credentials are successfully added, click **Edit credentials**  to view this information.

Your PE instance is now integrated with Continuous Delivery for PE.

To enable impact analysis for this instance, see Configure impact analysis.

Create environment node groups

In order for code deployments managed by Continuous Delivery for PE to work correctly, your environment node groups should be set up in a specific hierarchy.

Continuous Delivery for PE deploys changes to environment node groups. By setting up environment node groups, you define the groups of nodes that you can choose to deploy changes to.

1. In the PE console, click **Classification**.
2. If your node classification does not already include an All Environments node group, create one.

   > **Note:** The All Environments node group is added automatically in new installations of PE version 2018.1.5 and newer. If you're running an older version or have upgraded your PE instance, you must create this node group yourself.

   Click **Add group...** and create a new node group with the following specifications:

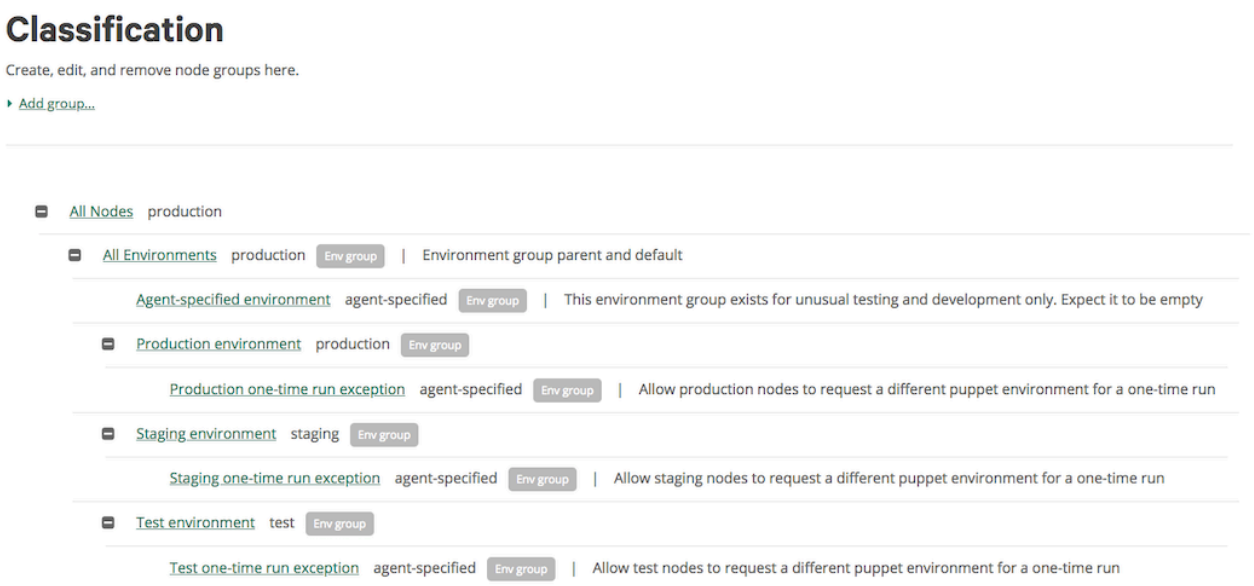
   - Parent name: All Nodes
   - Group name: All Environments
   - Environment: production
   - Environment group: yes
   - Description: Environment group parent and default

3. Open the All Environments node group and add a new rule:
   - Fact: name
   - Operator: ~
   - Value: `.*`
4. Edit the Agent-specified environment node group so that All Environments is its parent. This group should have no rules, and won't match any nodes.
5. Edit the Production environment node group so that All Environments is its parent. If necessary, modify its rules so that it matches only the correct nodes.
6. For each of your environment groups (such as testing, staging, and production), create an environment node group.
   a) Create a Git branch to represent the environment.
   b) Run `puppet code deploy <ENVIRONMENT_NAME>`.
   c) Create a new environment node group with the following specifications:
      - Parent name: All Environments
      - Group name: <ENVIRONMENT_NAME>
      - Environment: <ENVIRONMENT>
      - Environment group: Yes
   d) Associate the relevant nodes with the environment group by creating rules or pinning nodes.

      Best practices for associating nodes with environment node groups:
      - Use the `pp_environment` trusted fact, or a similar custom fact, to define which environment each node belongs to. Write a rule in each environment group that uses `pp_environment` or your custom fact to match nodes.
      - See if other facts or trusted facts can be used to create rules that match nodes to one and only one environment group
      - If trusted facts, custom facts, or other facts cannot be used to determine node environments, use pinning. Pin each node to only one environment group.
   e) Specify the Git branch corresponding to the environment.
7. Optional: For each of your newly created environment groups, create a child environment group. Nodes from the parent environment group are allowed to drop into this exception group to test code from Git feature branches. Give each child environment group the following specifications:
   - Parent name: All <ENVIRONMENT>
   - Group name: <ENVIRONMENT> one-time run exception
   - Environment: Agent-specified
   - Environment group: Yes
   - Description: Allow <ENVIRONMENT> nodes to request a different Puppet environment for a one-time run

   Once the child environment node group is set up, give it the rule `(agent_specified_environment ~ .+)`. Do not pin any nodes to this node group.

The resulting environment node groups has a format similar to this:



Now that your environment nodes groups are configured, we can deploy new code to your nodes.

Configure impact analysis

Impact analysis is a Continuous Delivery for Puppet Enterprise (PE) tool that lets you see the potential impact that new Puppet code will have on your PE-managed infrastructure, even before the new code is merged. When you add impact analysis to a control repo's pipeline, Continuous Delivery for PE automatically generates a report on every proposed code change to that control repo.

See Analyzing the impact of code changes for more on impact analysis.

- Configure impact analysis on page 52

If you have integrated your PE instance with Continuous Delivery for PE, impact analysis was automatically set up for you during the integration. Make one classification change to start using impact analysis.

Configure impact analysis

If you have integrated your PE instance with Continuous Delivery for PE, impact analysis was automatically set up for you during the integration. Make one classification change to start using impact analysis.

Before you begin

Make sure you're ready to get started by completing the following:

- Install Continuous Delivery for PE.
- Integrate with your source control system.
- Integrate with Puppet Enterprise.

1. In the PE console, click **Classification** and open the **PE Infrastructure** group.
2. Select the **PE Master** group and click **Configuration**.
3. In the **Add new class** field, select **cd4pe::impact_analysis** and click **Add class**, then commit your change.

   If you don't find **cd4pe::impact_analysis** in the class list, click **Refresh** to update class definitions.
4. Run Puppet on the nodes in the **PE Master** group.

Important: This Puppet run restarts the pe-puppetserver service.

5. Optional: Set the maximum number of concurrent node catalog compiles allowed for each workspace. By default, 10 concurrent catalog compilations are allowed.

   Adjust this setting to manage the catalog compilation load placed on a PE instance by Continuous Delivery for PE impact analysis report generation.

   a) In the Continuous Delivery for PE web UI, click **Settings**.
   b) Click **Puppet Enterprise** and locate the PE instance you've configured to use impact analysis. Click **Edit credentials**.
   c) Locate the **Impact analysis credentials** section of the window. In the **Max concurrent catalog compiles** field, set the number of catalog compilations that are allowed to run simultaneously on this PE instance.
   d) Click **Save changes**.

Integrate with source control

Integrate your source control system with Continuous Delivery for Puppet Enterprise (PE) by following the appropriate set of instructions on this page.

Status notification prefixes for source control

Once integration between your Continuous Delivery for PE installation and your source control provider is complete, Continuous Delivery for PE sends information about the outcome of each stage of each pipeline run to your source control provider.

By default, Continuous Delivery for PE labels each pipeline stage as follows when reporting to your source control provider:

```
cd-pe/stage-<pipeline stage number>
```

This labeling system works just fine if you connect a control repo or module repo to one (and only one) workspace. But if more than one workspace is connected to a certain control repo or module repo, your source control system might receive identical notifications from multiple workspaces about multiple pipelines, and be unable to differentiate between them when performing automated testing.

To prevent this issue, you have the option of adding a status notification prefix to all the communications Continuous Delivery for PE sends from your workspace to your source control provider. By adding a status notification prefix, you ensure that your source control system is able to differentiate between and accurately act on pipeline status notifications coming from multiple workspaces to the same control repo or module repo.

To add a status notification prefix:

1. In the Continuous Delivery for PE web UI, click **Settings** > **Source control**.
2. In the **Status notification prefix** area of the page, click **Edit prefix**  .
3. Enter your chosen prefix. The name of your workspace is a good option. Click **Save**.

When you save your prefix, the example code updates to show prefixed pipeline status labels as they will be sent to your source control provider from this workspace.

Integrate with Azure DevOps Services

Continuous Delivery for PE works with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Create an Azure DevOps Services OAuth application in order to integrate your Azure DevOps Services instance with Continuous Delivery for PE and start using these tools.

> **Before you begin**
> An administrator on your team must create an Azure DevOps Services OAuth application for Continuous Delivery for PE.

Note: These instructions apply only to the Azure DevOps cloud offering. The hosted version, Azure DevOps Server, is not compatible with Continuous Delivery for PE.

1. Sign into Continuous Delivery for PE as the root user.
2. Click **Settings**, then click **Integrations**.

   Tip: The authorization callback URL required to create your OAuth app is shown in the root console.
3. Go to https://app.vsaex.visualstudio.com/app/register. Enter your company name.
4. In the **Application Information** section, enter a name for your OAuth application, such as `CD for PE`.
5. In the **Application website** field, enter the base URL for your Continuous Delivery for PE instance.
6. In the **Authorization callback URL** field, enter the authorization callback URL printed in the root console.
7. In the **Authorized scopes** section, select **Code (read and write)**.
8. Click **Create Application**. Your new application is created, and a new page showing the application's settings is displayed.

   Important: Leave this page open. You'll need the application settings information in the next step.
9. Return to the Continuous Delivery for PE root console. On the **Integrations** page, enter the application ID and client secret for your Azure DevOps Services OAuth application and click **Add**.

Once an Azure DevOps Services OAuth application is established for your organization, each workspace must be authenticated with the application in order to integrate the Continuous Delivery for PE instance with Azure DevOps Services. This process involves granting code read and write permissions and adding a public SSH key, which enables cloning of modules and control repos during automated tasks.

Important: If your organization uses Azure DevOps Services branch permissions to limit user access to Git branches, review the permissions granted to Continuous Delivery for PE users and ensure that these users can force push to the relevant control repos and module repos.

Important: Azure DevOps Services only supports cloning over SSH. HTTP(S) cloning is not supported. To use Azure DevOps Services, SSL must be enabled on Continuous Delivery for PE.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Source control**, then click **Azure DevOps**.
3. Click **Add credentials** to give Continuous Delivery for PE code read and write permissions for your Azure DevOps Services account. You are directed to a Microsoft page.
4. Click **Accept**. You are directed back to the **Source control** page.
5. Next, add the SSH key. Still in the Continuous Delivery for PE web UI, click **SSH key**.
6. Click **Show** to display your public SSH key. Click **Copy**.
7. In the Azure DevOps Services web UI, open the user menu and click **Security**, then click **SSH public keys**.
8. Click **Add** and paste your public SSH key into the **Key Data** field. Add a description and click **Save**.

Integrate with Bitbucket Cloud

Continuous Delivery for PE works with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Create a Bitbucket Cloud OAuth application in order to integrate your Bitbucket Cloud instance with Continuous Delivery for PE and start using these tools.

Before you begin

An administrator on your team must create a Bitbucket Cloud OAuth consumer for Continuous Delivery for PE.

1. Sign into Continuous Delivery for PE as the root user.

2. Click **Settings**, then click **Integrations**.

> **Tip:** The authorization callback URL required to create your OAuth consumer is shown in the root console.

3. In your organization's Bitbucket Cloud account, create an OAuth consumer. See Create a consumer in the Bitbucket Cloud documentation for instructions.

   Give the OAuth consumer the following permissions:

Category	Permissions
Account	Email, Read
Team membership	Read
Repositories	Read, Write
Pull requests	Read, Write
Webhooks	Read and write

4. When your OAuth application is created, note the key and secret shown on the OAuth settings page in the Bitbucket Cloud web UI.
5. Return to the Continuous Delivery for PE root console. On the **Integrations** page, enter the client ID (key) and client secret for your Bitbucket Cloud OAuth consumer and click **Add**.

Once a Bitbucket Cloud OAuth application is established for your organization, each workspace must be authenticated with the application in order to integrate the Continuous Delivery for PE instance with Bitbucket Cloud.

> **Important:** If your organization uses Bitbucket Cloud branch permissions to limit user access to Git branches, review the permissions granted to Continuous Delivery for PE users and ensure that these users have write access and the ability to rewrite history on the relevant control repos and module repos.

> **Note:** Bitbucket Cloud only supports cloning over HTTP(S). SSH cloning is not supported.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Source control**, then click **Bitbucket Cloud**.
3. Click **Add credentials** to give Continuous Delivery for PE code read and write permissions for your Bitbucket Cloud account.
4. Click **Add credentials**.

   At this point you'll be redirected to Bitbucket Cloud to authorize the OAuth application set up by your workspace administrator.

   Give Continuous Delivery for PE the following permissions on your Bitbucket Cloud account:

   - Access organizations, teams, and membership (read-only)
   - Access user email addresses (read-only)
   - Access public and private repositories

5. Click **Authorize application**.

Integrate with Bitbucket Server

Continuous Delivery for PEworks with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Integrate your Bitbucket Server instance with Continuous Delivery for PE in order to start using these tools.

> **Important:** If your organization uses Bitbucket Server branch permissions to limit user access to Git branches, review the permissions granted to Continuous Delivery for PE users and create an exemption rule that ensures these users can force push to the relevant control repos and module repos.

Note: Bitbucket Server only supports cloning over SSH. HTTP(S) cloning is not supported.

Note: Continuous Delivery for PE supports Bitbucket Server 5.0 and newer versions.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Source control**, then click **Bitbucket Server**.
3. In the **Bitbucket Server host** field, enter the public IP or DNS for your Bitbucket Server instance.
4. In the **Username** and **Password** fields, enter the credentials associated with the account you wish to connect to Continuous Delivery for PE.
5. In the **SSH port** field, enter the port number on which your Bitbucket Server listens for SSH requests. To locate this port number:
   a. In the Bitbucket Server web UI, click **Administration** (the gear icon) and then click **Server settings**.
   b. Locate the SSH port in the **SSH access** section of the **Server settings** page.
6. Optional: Enter the SSH base URL for your Bitbucket Server if it is different from the host URL. To view your SSH base URL:
   a. In the Bitbucket Server web UI, click **Administration** (the gear icon) and then click **Server settings**.
   b. Locate the SSH base URL in the **SSH access** section of the **Server settings** page.
7. Optional: Enter the SSH user for clones if it is something other than "git."
8. Click **Add credentials**.

Integrate with GitHub

Continuous Delivery for PE works with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Create a GitHub OAuth application in order to integrate your GitHub instance with Continuous Delivery for PE and start using these tools.

Before you begin

An administrator on your team must create a GitHub OAuth application for Continuous Delivery for PE.

1. Sign into Continuous Delivery for PE as the root user.
2. Click **Settings**, then click **Integrations**.
3. In your organization's GitHub account, create an OAuth application. See Creating an OAuth App in the GitHub documentation for instructions.

   Tip: In the **Homepage URL** field, enter the base URL for your Continuous Delivery for PE instance (`http://<CD4PE-HOST-SERVER>:8080`). The **Authorization callback URL** is shown in the Continuous Delivery for PE root console.
4. Once your OAuth application is created, note the Client ID and Client Secret shown on the application's page in the GitHub UI.
5. Return to the Continuous Delivery for PE root console. On the **Integrations** page, enter the client ID and secret for your GitHub OAuth application and click **Add**.

Once a GitHub OAuth application is established for your organization, each workspace must be authenticated with the application in order to integrate the Continuous Delivery for PE instance with GitHub.

Important: If your organization uses protected branches on GitHub, make sure that force pushing is allowed to protected branches, or that the GitHub Administrator user is used when connecting a control repo or module repo to Continuous Delivery for PE.

Note: GitHub only supports cloning over HTTP(S). SSH cloning is not supported.

1. In the Continuous Delivery for PE web UI, click **Settings**.

2. Click **Source control**, then click **GitHub**.
3. Click **Add credentials**.

   At this point you'll be redirected to GitHub to authorize the OAuth application set up by your team's administrator.

   Give Continuous Delivery for PE the following permissions on your GitHub account:

   - Access organizations, teams, and membership (read-only)
   - Access user email addresses (read-only)
   - Access public and private repositories
4. Click **Authorize application**.

Integrate with GitHub Enterprise

Continuous Delivery for PE works with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Integrate your GitHub Enterprise instance with Continuous Delivery for PE in order to start using these tools.

Important: If your organization uses protected branches on GitHub Enterprise, make sure that make sure that force pushing is allowed to protected branches, or that the GitHub Enterprise Administrator user is used when connecting a control repo or module repo to Continuous Delivery for PE.

Note: GitHub Enterprise only supports cloning over HTTP(S), SSH cloning is not supported.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Source control**, then click **GitHub Enterprise**.
3. In the **Host** field, enter the public IP or DNS for your GitHub Enterprise instance.
4. Create a token allowing Continuous Delivery for PE to access your GitHub Enterprise instance.
   a) In the GitHub Enterprise web UI, click your profile photo, then click **Settings** > **Developer settings**.
   b) Click **Personal access tokens**. Click **Generate new token**.
   c) Enter a token description, such as `CD for PE`.
   d) Select the **repo**, **read:org**, and **user:email** scopes.
   e) Click **Generate token**.
   f) Copy the personal access token created by GitHub Enterprise.
5. In the Continuous Delivery for PE web UI, enter the GitHub Enterprise token in the **Token** field.
6. Based on your GitHub Enterprise configuration, select either **This instance uses a standard CA certificate** or **This instance uses a custom CA certificate**. If you're using a custom certificate, paste the certificate in full in the **Custom CA certificate** field.
7. Click **Add credentials**.

Integrate with GitLab

Continuous Delivery for PE works with your existing source control system to track changes to your Puppet code and manage code deployments to your nodes. Integrate your GitLab instance with Continuous Delivery for PE in order to start using these tools.

Important: If your organization uses protected branches on GitLab, make sure that the GitLab user account connected to Continuous Delivery for PE is assigned to a GitLab role with “allow” rules that enable the user to push to the protected branch.

Note: GitLab supports cloning over both SSH and HTTP(S). The cloning protocol is set per Continuous Delivery for PE workspace.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. Click **Source control**, and then click **GitLab**.

3. In the **Host** field, enter the public IP or DNS for your GitLab instance.
4. Create a token allowing Continuous Delivery for PE to access your GitLab instance.
   a) In the GitLab web UI, navigate to your user settings and click **Access tokens**.
   b) Enter a name for the application, such as `CD for PE`, and set an expiration date for the token.
   c) Select the **api** and **read_user** scopes.
   d) Click **Create personal access token**.
   e) Copy the personal access token created by GitLab.
5. In the Continuous Delivery for PE web UI, enter the GitLab token in the **Token** field.
6. Select whether your workspace will clone GitLab repositories via SSH or HTTP(S).
   a) For SSH:
      - **Optional:** Add the SSH user's credentials in the **SSH user** field.
      - **Optional:** In the **SSH port** field, specify the port on which your GitLab server listens for SSH requests. The default port number is 22.
   b) For HTTP(S):
      - If you're using a custom certificate, paste the certificate in full into the **Custom CA certificate** field.
7. Click **Add credentials**.

Configure job hardware

Job hardware, or the servers Continuous Delivery for Puppet Enterprise (PE) uses to run tests on your Puppet code, must be configured before you can begin running tests or using pipelines.

Configure job hardware running a Puppet agent

Job hardware, or the servers Continuous Delivery for PE uses to test your Puppet code, must be configured before you begin running jobs or using pipelines. Any node with a Puppet agent installed can be designated as job hardware.

Before you begin
Review the sizing guide in Job hardware requirements.

Important: Only the administrator of a workspace or a super user can configure job hardware for a workspace.

1. Install a Puppet agent on each of the nodes you wish to use as job hardware. See Installing agents in the PE documentation for details.
2. Make sure your Continuous Delivery user role in PE includes the permission to run the `cd4pe_jobs::run_cd4pe_job` task.
3. Install the `puppetlabs-cd4pe_jobs` module, which is required to run Continuous Delivery for PE jobs on your nodes:
   a) Add the `puppetlabs-cd4pe_jobs` module to the following:
      - The Puppetfile for the production environment on the PE master that manages the agent nodes you've selected as job hardware
      - The Puppetfile on the `master` branch of the control repo associated with the PE master that manages the agent nodes you've selected as job hardware

      A sample Puppetfile entry:

      ```
      mod 'puppetlabs-cd4pe_jobs', '1.5.0'
      ```

   b) Deploy the updated code to the production environment:

      ```
      puppet code deploy production --wait
      ```

4. Install and configure Docker on each node. Docker is required for the Docker-based pre-built jobs included in Continuous Delivery for PE. See the Installing Docker instructions for details.
5. Finally, tell Continuous Delivery for PE that the nodes are ready to be used as job hardware for Docker-based jobs by assigning them to the **Docker** hardware capability. Capabilities organize your job hardware servers and ensure that jobs run on hardware with the right characteristics. Continuous Delivery for PE automatically creates a **Docker** hardware capability for you.
   a) In the Continuous Delivery for PE web UI, click **Hardware**.
   b) Locate the **Docker** capability and click **+ Edit**.
   c) Select the PE instance that manages the nodes you've selected as job hardware. Then, select your job hardware nodes.

      The selected nodes are added to the **Hardware with this capability** list on the right.
   d) Click **Save**.

Your job hardware is now configured and ready for use.

Tip: If you're managing a large set of job hardware nodes, you can create additional capabilities to distribute the testing load and ensure that each job runs on a server with the correct characteristics. See Add job hardware capabilities for instructions.

Note: Due to the fact that Continuous Delivery for PE job hardware nodes are classified as PE infrastructure nodes, the `puppet_agent` module cannot be used to successfully upgrade Puppet agents running on job hardware. To upgrade the Puppet agent on a job hardware node, use the agent upgrade script.

Add job hardware capabilities

Job hardware servers are organized by capabilities in Continuous Delivery for PE. A capability is a tag that indicates what type of jobs can run on that job hardware server. If you're managing a large fleet of job hardware servers, use capabilities to distribute the testing load.

Capabilities organize your job hardware servers and ensure that jobs run on hardware with the right characteristics. For example, for a new job that requires the use of Python 3, create a capability called **python-3** in Continuous Delivery for PE and assign job hardware servers that have Python 3 installed and configured to that capability. When creating the new job that uses Python 3, indicate that the job requires job hardware with the **python-3** capability. When running that job, Continuous Delivery for PE will automatically locate a job hardware server with the **python-3** capability, and will use that job hardware server to run the job.

In addition to software requirements, useful capabilities include:

- Operating systems
- Organizational designations, such as dev-team-philly-testing
- Puppet testing resources, such as onceover or Puppet Development Kit

Important: Only the administrator of a workspace, the root user, and super users can add hardware capabilities to a workspace.

Note: Continuous Delivery for PE automatically creates a **Docker** capability for you.

1. In the Continuous Delivery for PE web UI, click **Hardware**.
2. To create a new capability, click **+ Add capability**. Give your new capability a name.
3. Next, assign job hardware servers to the capability. Select the PE instance that manages the node you've selected as job hardware. Then, select the nodes you wish to assign to the capability.

   The selected nodes are added to the **Hardware with this capability** list on the right.
4. When you've selected all the nodes you want to assign to the capability, click **Save**.

Your job hardware is now assigned to a capability. You can add this capability to the jobs in your workspace by clicking **Jobs** > **Edit job** and selecting from the list of capabilities.

Configure global shared job hardware running a Puppet agent

Global shared job hardware can be used by all workspaces in your Continuous Delivery for PE installation. The root user or a super user must set up these special job hardware servers in the root console.

Before you begin
Review the sizing guide in Job hardware requirements.

1. Install a Puppet agent on each of the nodes you wish to use as global shared job hardware. See Installing agents in the PE documentation for details.
2. Make sure your Continuous Delivery user role in PE includes the permission to run the `cd4pe_jobs::run_cd4pe_job` task.
3. Install the `puppetlabs-cd4pe_jobs` module, which is required to run Continuous Delivery for PE jobs on your nodes:
   a) Add the `puppetlabs-cd4pe_jobs` module to the following:
      - The Puppetfile for the production environment on the PE master that manages the agent nodes you've selected as job hardware
      - The Puppetfile on the `master` branch of the control repo associated with the PE master that manages the agent nodes you've selected as job hardware

      A sample Puppetfile entry:

      ```
      mod 'puppetlabs-cd4pe_jobs', '1.5.0'
      ```

   b) Deploy the updated code to the production environment:

      ```
      puppet code deploy production --wait
      ```
4. Install and configure Docker on your selected global shared job hardware nodes. See the Installing Docker instructions for details.
5. In the root console, click **Hardware**.
6. All global shared job hardware servers must be assigned to the **Docker** capability, which is automatically created for you. Locate the **Docker** capability and click **+ Edit**.
7. Next, assign all your global shared job hardware servers to the **Docker** capability. Select the PE instance that manages the node you've selected as job hardware. Then, select the global shared job hardware nodes.

   The selected nodes are added to the **Hardware with this capability** list on the right.

8. When you've selected all the nodes you want to assign to the capability, click **Save**.
9. As needed, create additional capabilities for your global job hardware servers. To create a new capability, click **+ Add capability** and give your new capability a name, then follow steps 6 and 7.

> **Tip:** To learn more about capabilities, see Add job hardware capabilities.

Your global shared job hardware is now configured. Users in all workspaces will now see a **Use shared hardware** option when creating or editing a job. Jobs that are configured to run on your global shared job hardware have the **Docker** capability automatically selected, and will run on the global shared job hardware assigned to the **Docker** capability.

> **Note:** Due to the fact that Continuous Delivery for PE job hardware nodes are classified as PE infrastructure nodes, the `puppet_agent` module cannot be used to successfully upgrade Puppet agents running on job hardware. To upgrade the Puppet agent on a job hardware node, use the agent upgrade script.

Migrate job hardware

The Continuous Delivery agent is deprecated as of Continuous Delivery for PE version 3.4.0, and will be removed in a future release. The Continuous Delivery agent will continue to work until the removal date, but you can migrate your job hardware to use a Puppet agent at any time.

To migrate your job hardware from using the Continuous Delivery agent to using a Puppet agent, first uninstall the Continuous Delivery agent, then install a Puppet agent and the `puppetlabs-cd4pe_jobs` module.

> **Note:** These migration instructions apply to both job hardware associated with a single Continuous Delivery for PE workspace and global shared job hardware. The workspace administrator or a super user must complete the migration.

1. SSH into your job hardware agent node and run the following:

```
sudo /usr/local/bin/distelli agent stop
sudo /usr/local/bin/distelli agent uninstall
```

2. In the Continuous Delivery for PE web UI (or in the root console if you're migrating a global shared job hardware server), click **Hardware**. Locate the job hardware agent node in the list of servers and click **Remove hardware node**.
3. Install a Puppet agent on each of the nodes you wish to use as job hardware. See Installing agents in the PE documentation for details.
4. Make sure your Continuous Delivery user role in PE includes the permission to run the `cd4pe_jobs::run_cd4pe_job` task.
5. Install the `puppetlabs-cd4pe_jobs` module, which is required to run Continuous Delivery for PE jobs on your nodes:
   a) Add the `puppetlabs-cd4pe_jobs` module to the following:
      - The Puppetfile for the production environment on the PE master that manages the agent nodes you've selected as job hardware
      - The Puppetfile on the `master` branch of the control repo associated with the PE master that manages the agent nodes you've selected as job hardware

      A sample Puppetfile entry:

```
mod 'puppetlabs-cd4pe_jobs', '1.5.0'
```

   b) Deploy the updated code to the production environment:

```
puppet code deploy production --wait
```

6. Confirm that Docker is installed and configured on each node. Docker is required for the Docker-based pre-built jobs included in Continuous Delivery for PE. See the Installing Docker instructions for details.
7. Finally, tell Continuous Delivery for PE that the nodes are ready to be used as job hardware for Docker-based jobs by assigning them to the **Docker** hardware capability. Capabilities organize your job hardware servers and ensure that jobs run on hardware with the right characteristics. Continuous Delivery for PE automatically creates a **Docker** hardware capability for you.
   a) In the Continuous Delivery for PE web UI, click **Hardware**.
   b) Locate the **Docker** capability and click **+ Edit**.
   c) Select the PE instance that manages the nodes you've selected as job hardware. Then, select your job hardware nodes.

      The selected nodes are added to the **Hardware with this capability** list on the right.
   d) Click **Save**.

Your job hardware is now migrated and ready for use.

> **Tip:** If you're managing a large set of job hardware nodes, you can create additional capabilities to distribute the testing load and ensure that each job runs on a server with the correct characteristics. See Add job hardware capabilities for instructions.

Continuous Delivery agent

DEPRECATED: The Continuous Delivery agent is deprecated as of Continuous Delivery for PE version 3.4.0, and will be removed in a future release. While existing job hardware running the Continuous Delivery agent will continue to function until the removal date, you cannot make changes or updates to these job hardware servers.

> **Important:** See Migrating your job hardware agents for instructions on updating your job hardware to use a Puppet agent.

Selecting a job hardware installation method

There are two methods for authenticating your account when configuring job hardware to use the Continuous Delivery agent. Which you choose to employ is mainly a question of how many servers you're designating as job hardware.

When installing the Continuous Delivery agent, you must enter the email address and password associated with your Continuous Delivery for PE account. Doing this manually is fine if you're setting up a server or two, but if you're configuring a larger number of servers, adding these credentials can quickly become cumbersome. In this case, using a `distelli.yml` file to automatically pass your credentials to the agent is the more efficient method.

Configure job hardware with the web UI

To designate a server as job hardware, install the Continuous Delivery agent on the server and mark it as active job hardware in the web UI.

> **Tip:** This method is best to use if you're configuring a small number of servers.

1. In the Continuous Delivery for PE web UI, click **Hardware**.
2. Click **Add job hardware**.
3. Select **Linux / MacOS** or **Windows**. Install the Continuous Delivery agent by running the commands shown in the web UI on the machine you've designated as job hardware.

   > **Important:** On Linux hosts, the agent installation process adds a sudoers rule to `/etc/sudoers.d/distelli` that enables the `distelli` user to run any command with sudo privileges and without requiring a password.

4. When prompted for your email and password, enter the credentials associated with your Continuous Delivery for PE account.

   Important: Do not enter the root account credentials.

5. At the prompt `To which hardware collection should this agent be added?` select the relevant workspace.
6. In the Continuous Delivery for PE web UI, close the **Add job hardware** pane. You might need to refresh the **Job hardware** page to see your newly configured job hardware.
7. Click the **Job hardware active** toggle to mark your new job hardware as active.
8. Optional: Click **+ Add capability** and enter up to three capabilities associated with this piece of job hardware, such as `PUPPET-AGENT` or `DOCKER`, pressing **Save** after each addition.

   The Continuous Delivery agent automatically detects and displays four reserved capabilities: **WINDOWS**, **LINUX**, **DARWIN**, and **RASPBIAN**.

   When creating a job, you can specify which job hardware capabilities are required. This allows you to ensure that jobs requiring specific conditions, such as a particular operating system or piece of software, are run only on job hardware meeting those conditions.

   Important: In order to run jobs in Docker containers (including pre-built jobs), make sure Docker is installed on your job hardware and set **DOCKER** as a capability.

   Note: Running Docker-enabled job hardware on an Alpine Linux base image requires the libgcc, bash, wget, and ca-certificates packages.

Configure job hardware with `distelli.yml`

By setting up a `distelli.yml` file containing your user-specific agent credentials, you can configure job hardware without manually entering your account credentials.

Tip: This method is best to use if you're configuring a large number of servers and don't want to enter your credentials manually each time.

1. Create your agent credentials.
   a) In the Continuous Delivery for PE web UI, click **Settings**.
   b) Click **Hardware agents**, then click **Create agent credentials**.
   c) Continuous Delivery for PE creates an access token and secret key for your account. Click **Show** to display your secret key. Leave this page open while you set up your `distelli.yml` file.
2. Create a file named `distelli.yml` and store it in a convenient location.
3. Add the following to your `distelli.yml` file, pasting in the access token and secret key you generated in Step 1:

   ```
   DistelliAccessToken: <MY_ACCESS_TOKEN>
   DistelliSecretKey: <MY_SECRET_KEY>
   ```

   Save the file and exit.
4. In the Continuous Delivery for PE web UI, click **Hardware**.
5. Click **Add job hardware**.

6. Select **Linux / MacOS** or **Windows**. SSH into the machine you've chosen to designate as job hardware. In sudo or Administrator mode, install the Continuous Delivery agent by running the commands shown in the web UI, adding `-conf <PATH_TO_DISTELLI.YML_FILE>` to the end of the `distelli agent install` command.
   For example, `/usr/local/bin/distelli agent install -conf <PATH_TO_DISTELLI.YML_FILE>`.

   > **Important:** On Linux hosts, the agent installation process adds a sudoers rule to `/etc/sudoers.d/distelli` that enables the `distelli` user to run any command with sudo privileges and without requiring a password.

7. At the prompt `To which hardware collection should this agent be added?` select the relevant workspace.
8. In the Continuous Delivery for PE web UI, click **Hardware**.
9. Click the **Job hardware active** toggle to mark your new job hardware as active.
10. Optional: Click **+ Add capability** and enter up to three capabilities associated with this piece of job hardware, such as `PUPPET-AGENT` or `DOCKER`, clicking **Save** after each addition.

    The Continuous Delivery agent automatically detects and displays four reserved capabilities: **WINDOWS**, **LINUX**, **DARWIN**, and **RASPBIAN**.

    When creating a job, you can specify which job hardware capabilities are required. This allows you to ensure that jobs requiring specific conditions, such as a particular operating system or piece of software, are run only on job hardware meeting those conditions.

    > **Important:** In order to run jobs in Docker containers (including pre-built jobs), make sure Docker is installed on your job hardware and set **DOCKER** as a capability.

    > **Note:** Running Docker-enabled job hardware on an Alpine Linux base image requires the libgcc, bash, wget, and ca-certificates packages.

Upgrade the Continuous Delivery agent

To upgrade the Continuous Delivery agent to a more recent version, reinstall the agent on top of the existing agent.

> **Important:** The Continuous Delivery agent is deprecated as of Continuous Delivery for PE version 3.4.0, and will be removed in a future release. Upgrading the Continuous Delivery agent is no longer supported. See Migrating your job hardware agents for instructions on updating your job hardware to use a Puppet agent.

1. In the Continuous Delivery for PE web UI, click **Hardware**.
2. Click **Add job hardware**.
3. Select **Linux / MacOS** or **Windows**. SSH into the machine you're using as job hardware. In sudo or Administrator mode, install the Continuous Delivery agent by running the commands shown in the web UI.

   > **Important:** If you created a distelli.yml file to store your agent credentials, add `-conf <PATH_TO_DISTELLI.YML_FILE>` to the end of the `distelli agent install` command and skip to step 5 below.

4. When prompted for your email and password, enter the credentials associated with your Continuous Delivery for PE account.

   > **Important:** Do not enter the root account credentials.

5. In the Continuous Delivery for PE web UI, click **Hardware** and locate the newly upgraded server.
6. Click the **Job hardware active** toggle to mark the upgraded job hardware as active.
7. Click **+ Add capability** and enter up to three capabilities associated with this piece of job hardware, such as `PUPPET-AGENT` or `DOCKER`, clicking **Save** after each addition.

Configure global shared job hardware running a Continuous Delivery agent

The Continuous Delivery agent is deprecated as of Continuous Delivery for PE version 3.4.0, and will be removed in a future release. While existing global shared job hardware running the Continuous Delivery agent will continue to function until the removal date, you cannot make changes or updates to these shared job hardware servers.

A super user must perform this action. Once set up, all super users and the root user can view global shared job hardware in the root console.

> **Important:** See Migrating your job hardware agents for instructions on updating your global shared job hardware to use a Puppet agent.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar.
2. Click **Hardware**, then click **Add job hardware**.
3. Select **Linux / MacOS** or **Windows**. Install the Continuous Delivery agent by running the commands shown in the web UI on the machine you've designated as global shared job hardware.

   > **Important:** On Linux hosts, the agent installation process adds a sudoers rule to `/etc/sudoers.d/distelli` that enables the `distelli` user to run any command with sudo privileges and without requiring a password.

4. When prompted for your login email and password, enter the credentials associated with your super user account.
5. At the prompt `To which hardware collection should this agent be added?` select `Global shared hardware`.
6. In the root console, close the **Add job hardware** pane. You might need to refresh the **Job hardware** page to see your newly configured global shared job hardware.
7. Click the **Job hardware active** toggle to mark your new job hardware as active.
8. Optional: Click **+ Add capability** and enter up to three capabilities associated with this piece of job hardware, such as `PUPPET-AGENT` or `DOCKER`, pressing **Save** after each addition.

   The Continuous Delivery agent automatically detects and displays four reserved capabilities: **WINDOWS**, **LINUX**, **DARWIN**, and **RASPBIAN**.

   When creating a job, users can specify which job hardware capabilities are required. This ensures that jobs requiring specific conditions, such as a particular operating system or piece of software, are run only on job hardware meeting those conditions.

   > **Important:** In order to run jobs in Docker containers (including pre-built jobs), make sure Docker is installed on your job hardware and set **DOCKER** as a capability.

   > **Note:** Running Docker-enabled job hardware on an Alpine Linux base image requires the libgcc, bash, wget, and ca-certificates packages.

Now that shared global job hardware is available, a **Use shared hardware** option is shown when creating or editing a job.

Configure LDAP

Continuous Delivery for Puppet Enterprise (PE) supports use of the Lightweight Directory Access Protocol (LDAP) for managing user authentication. Once an LDAP configuration is in place, use group mapping to associate your existing LDAP groups with role-based access control (RBAC) groups in Continuous Delivery for PE.

For organizational or failover protection purposes, you can add multiple LDAP configurations, each specifying a separate LDAP server, to your Continuous Delivery for PE instance. Continuous Delivery for PE uses the LDAP configurations you set up to search your LDAP users in a specified order. Once a user is found, the search ends and that LDAP configuration is used to perform the login operation.

Configuring LDAP server search result limits

Beginning with version 3.10.0, by default Continuous Delivery for PE requests 500 search results at a time from a connected LDAP server. If your LDAP server has a search result limitation below 500, you can configure Continuous Delivery for PE to match the LDAP server's search result threshold by setting the `CD4PE_LDAP_GROUP_SEARCH_SIZE_LIMIT` environment variable on the `cd4pe_docker_extra_params` parameter. See Advanced configuration options for more information.

Create a new LDAP configuration

Add an LDAP configuration to Continuous Delivery for PE by providing key information on the mapping of user and group attributes in your LDAP server implementation.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Settings**, then click **Single sign on**.
3. Select **LDAP** and click **+ Add LDAP configuration**.

4. In the **New LDAP configuration** pane, enter the requested information as per the instructions below.

   Name

   A friendly identifier for this LDAP configuration. This name cannot be changed, so select it carefully.

   Endpoint URL

   The endpoint URL of the LDAP server, including the LDAP scheme, hostname, and port. If these aren't included, the default scheme is `ldaps` and the default port is 636.

   Bind DN

   The distinguished name of the LDAP account that Continuous Delivery for PE will bind as when performing LDAP operations. An admin account or a service account created specifically for Continuous Delivery for PE is generally used here. If you choose to create a new account, ensure that it has permission to search for users and groups.

   Bind DN password

   The password associated with the bind DN account.

   > **Note:** You must enter this password each time the LDAP configuration is updated. Entering the password is not required when disabling the LDAP configuration.

   User base DN

   The LDAP base DN that informs Continuous Delivery for PE where users are located in the directory. The more specific the DN, the better your LDAP search performance will be.

   User attribute

   Which LDAP user attribute to use to map LDAP users to Continuous Delivery for PE usernames. The `mail` attribute is most commonly used, but any attribute can be used so long as duplicate values aren't present in the LDAP database.

   Optional: User base filter

   A filter that Continuous Delivery for PE can use to restrict user search results. This is useful in edge cases where certain users should be included or excluded.

   Group base DN

   The LDAP base DN that informs Continuous Delivery for PE where groups are located in the directory. The more specific the DN, the better your LDAP search performance will be.

   > **Note:** If users and groups are stored in the same location, the value in this field will be the same as the value in the **User attribute** field.

   Group user attribute

   The user attribute that group entries use to identify users. In most cases the value of this option is `dn`.

   Group member attribute

   The group attribute that maps to a group member. In most cases the value of this option is either `member` or `uniqueMember`.

   Group name attribute

   The group attribute that identifies the group name. In most cases the value of this option is `cn`.

   Optional: Group base filter

   A filter that Continuous Delivery for PE can use to restrict group search results. This is useful in edge cases where certain groups should be included or excluded.

   User object class

   Specifies the value of the `objectClass` attribute that allows Continuous Delivery for PE to query user entries. In most cases the value of this option is `user` or `person`.

   Group object class

   Specifies the value of the `objectClass` attribute that allows Continuous Delivery for PE to query group entries. In most cases the value of this option is `group` or `groupOfUniqueNames`.

   Optional: Mail attribute

   The LDAP user attribute used to identify each member's email address. Defaults to `mail` if unset.

5. Select the **Priority** number for this LDAP configuration. This number indicates the order in which your LDAP configurations are used to search for users during login and when synchronizing groups.
6. Optional: Enter the trusted server's CA certificate. If the LDAP server uses a certificate signed by a trusted CA, you do not need to enter a CA certificate here.
7. Optional: Set the toggle to enable recursive LDAP queries for nested groups. This option is available only if your LDAP server's implementation supports the ExtensibleMatch search filter.
8. Set the toggle to **Enable LDAP**, then click **Run configuration test** to check the connection between Continuous Delivery for PE and the LDAP server.

> **Note:** This configuration test checks to see if the LDAP server can be reached; it does not test the other configuration options.

9. Click **Save configuration**. Your new LDAP configuration is now shown on the **Single sign on** settings page, where you have the option to edit or delete the configuration.

> **Important:** Once you enable an LDAP configuration, Continuous Delivery for PE will automatically disable all local Continuous Delivery for PE accounts other than the root account, and will attempt to use LDAP authentication. If your LDAP authentication fails, sign in as the root user by adding `/root/login` to the end of the base URL of the Continuous Delivery for PE web UI and adjust your settings.

Create an LDAP group map

Once you add an LDAP configuration to Continuous Delivery for PE, use a group map to map your existing LDAP groups to Continuous Delivery for PE RBAC groups. This makes it possible to mirror group membership defined in LDAP to groups in Continuous Delivery for PE.

> **Before you begin**
> Add at least one LDAP configuration to your Continuous Delivery for PE instance.

1. Log into the root console by adding `/root/login` to the end of the base URL of the web UI and signing in as the root user.
2. Click **Settings**, then click **Single sign on**.
3. Click **LDAP**, then click **Manage groups**. Click **+ Add LDAP group mapping**
4. Select the LDAP configuration to associate the group mapping with.
5. From the list of available LDAP groups, select the group to be used to perform the mapping.

> **Tip:** You can search for groups by name (partial matches and case-insensitive matches allowed) or distinguished name (case-insensitive matches allowed).

6. Select a Continuous Delivery for PE account that is associated with the RBAC group you wish to map to the selected LDAP group.
7. From the list of available Continuous Delivery for PE RBAC groups, select the RBAC group you wish to map to the selected LDAP group.
8. Click **Add group mapping**.
   Once a group map is set up, Continuous Delivery for PE synchronizes with your LDAP groups based on the mapping you create.

Configure SMTP

Configure SMTP for your Continuous Delivery for Puppet Enterprise (PE) installation so that users can receive email notifications from the software.

The root user or a super user must complete this process. The email address associated with the root user is used as the **from:** address when email is sent by Continuous Delivery for PE.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Settings** > **SMTP**.
3. If your SMTP server requires authentication, enter the **SMTP username** and **SMTP password** in the relevant fields.

   CAUTION: The specified SMTP account must have permission to send emails from the email address associated with the Continuous Delivery for PE root user.
4. Enter the name of your **SMTP host** and the **SMTP port** number in the relevant fields.

Note: Contact your email server administrator if you need help determining the correct SMTP port.

5. If your server requires TLS authentication, click the toggle to **Enable TLS**.
6. Click **Save settings**.
7. To test your new SMTP configuration, generate a password reset email.
   a) Log out of Continuous Delivery for PE. On the sign-in screen, click **Forgot your password?**
   b) Enter the email address associated with your Continuous Delivery for PE account and click **Send reset instructions**.

   Important: You can safely ignore the reset password instructions and keep your current password.

Now that SMTP is configured, Continuous Delivery for PE sends email in these situations:

- Password reset instructions when requested by a user
- Password reset confirmation once a password is updated
- Deployment approval request notifications when a deployment to a protected environment is proposed

Configure SSL

Continuous Delivery for Puppet Enterprise (PE) supports the use of Secure Sockets Layer (SSL) for enhanced security when using the software.

When SSL is enabled, it impacts these elements of your Continuous Delivery for PE installation:

- The web UI
- Communication of Puppet agents running on job hardware nodes
- OAuth applications used with some source control providers

SSL configuration requirements and prerequisites

Before enabling SSL on your system, review the following important information.

1. Enabling SSL requires super user permissions.
2. Enabling SSL requires a restart of the Continuous Delivery for PE Docker container.
3. **If you installed Continuous Delivery for PE from the PE console:** Before configuring SSL, you must install the `cd4pe` module, which automates upgrades of Continuous Delivery for PE and manages your configuration. For instructions, see Automate upgrades of Continuous Delivery for PE.
4. **If you are running legacy (now deprecated) Continuous Delivery agents on your job hardware**: After configuring SSL you must delete and reinstall the Continuous Delivery agent on all job hardware. See **What to do next** at the bottom of this page for instructions.

Setting up a new SSL configuration

Configure your Continuous Delivery for PE instance to use SSL by entering the relevant certificates in the root console and then updating your web UI endpoint to reflect the new DNS host and SSL port.

Before you begin

Review the SSL configuration requirements and prerequisites section above.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Settings** and make sure you're viewing the **Endpoints** tab.
3. In the Configure SSL area, paste in the CA certificate, server certificate, and server private key for your Continuous Delivery for PE host.

   Note: If you also have an intermediary CA certificate, paste both the CA certificate and the intermediary CA certificate into the **CA certificate** field.
4. Optional: Click the toggle to **Enable SSL**.

   Note: You can leave your SSL configuration disabled and save the information you've entered. If SSL information is entered and saved but not enabled, your certificates are saved and the private key is saved in an encrypted format until you're ready to enable SSL.
5. Click **Save SSL settings**. If you've enabled SSL, proceed to the next step.
6. In the Configure Endpoints area of the page, update the web UI endpoint. The format for the new web UI endpoint is `https://<DNS_HOST>:<SSL_PORT>`.

   By default, Continuous Delivery for PE uses port 8443 for SSL.
7. **Azure DevOps Services users:** Update the backend service endpoint to use `https`. This change allows Azure DevOps Services webhooks to function correctly.

   Important: Continuous Delivery for PE does not support webhooks using SSL. This step is only to provide compatibility with Azure DevOps Services.
8. Click **Update endpoints**.
9. Stop and restart the Continuous Delivery for PE container by running the following:

   ```
   service docker-cd4pe stop
   service docker-cd4pe start
   ```

You can now access Continuous Delivery for PE over SSL by pointing your web browser to the new web UI endpoint you entered. Access over both `https` and `http` is allowed.

If you are running legacy (now deprecated) Continuous Delivery agents on your job hardware: After configuring SSL you must delete and reinstall the Continuous Delivery agent on all job hardware. To delete and reinstall an agent, SSH into your job hardware agent node and run the following:

```
sudo /usr/local/bin/distelli agent stop
sudo /usr/local/bin/distelli agent uninstall
sudo /usr/local/bin/distelli agent install
```

Note: The `uninstall` command throws an expected `POST not supported for resource / decommission-server/` error. You can safely ignore this error.

Managing teams and team members

Managing workspaces

Workspaces enable you to share access to key Continuous Delivery for PE resources, such as control repos, modules, and jobs, with the other members of your team. Once you set up a workspace, add your team members to that workspace and give them the user permissions needed to do their work.

Best practices when creating workspaces

This section offers suggestions and best practices for creating workspaces. This guidance is intended to help you and your organization understand Continuous Delivery for PE workspaces and use them effectively.

Workspaces support teams writing Puppet code and deploying that code to nodes managed by PE. Whether your organization tasks one team with writing and deploying all Puppet code, or has organized multiple teams to write and test Puppet code while a central deployment team pushes those changes to production, workspaces can help you ensure that each team member has exactly the Continuous Delivery for PE resources they need.

If your organization uses...	**Workspace recommendation**
One team to write, test, and deploy all Puppet code A **single source control repository** to store all Puppet code	• Use one workspace for the whole team • Set permissions carefully to ensure that each team member has access to the resources they need
Multiple teams to write and test Puppet code **Multiple source control repositories** to store Puppet code A **deployment team** responsible for getting Puppet code changes into production	• Set up separate workspaces for each writing and testing team, ideally one workspace for each control repository • Create a separate workspace for the deployment team, and allow these team members access to all other workspaces • Set permissions carefully to ensure that each team member has access to the resources they need

Set up a new workspace for a team

Perform these steps to set up a single workspace for a small, centralized team, or to create one of several team workspaces for a larger, less centralized organization.

Before you begin

Make sure all members of the team have created individual Continuous Delivery for PE accounts.

Note: When new users reach the **Choose a workspace** screen in the new account creation process, ask them to log out of Continuous Delivery for PE until the team workspace is fully set up.

1. Create a new workspace by navigating to **Manage workspaces** at top of the navigation bar in the Continuous Delivery for PE web UI and clicking **+ Add new workspace**. Give the workspace a descriptive name (if necessary, you can change this later in the **Settings** menu).

   When you create a workspace, you are automatically set as the owner of the workspace. Other Continuous Delivery for PE super users can also access your workspace by navigating to the root console and clicking the name of your workspace in the **Workspaces** tab.
2. Switch to the newly created workspace by clicking its name in the list of **My workspaces**, or by selecting it from the workspaces menu in the Continuous Delivery for PE web UI.
3. Follow the new workspace prompts at the top of the screen to set up the required resources for the workspace:
   a) If necessary, integrate Puppet Enterprise
   b) Integrate with your source control system
   c) Set up job hardware for this workspace or use global shared job hardware
4. Add the control repo where your team keeps its Puppet code to Continuous Delivery for PE. Click **Control repos**, then click **Add control repo**.
5. Add the members of your team as users of the workspace. Click **Settings**, then **Users**, and add each team member.
6. Click **Groups** and create one or more new groups, assigning appropriate permissions and users to each group. Make sure that every team member is assigned to at least one group.

   If a user is added to a workspace but not assigned to any groups, they will see a 403 error when they sign into Continuous Delivery for PE.
7. Invite the members of the team to sign into Continuous Delivery for PE. They'll now see your newly created team workspace in the workspaces area of the Continuous Delivery for PE. web UI.

Team members are now able to view and interact with the resources you've added to the workspace according to the permissions you've assigned.

You can delete unneeded workspaces by navigating to **Settings** > **Workspace** and clicking **Delete workspace**. Only the workspace owner or a super user can perform this action.

Transfer ownership of a workspace

When the owner of a workspace changes teams, leaves your organization, or is otherwise no longer the right person to manage a workspace, you can reassign ownership of the workspace to a different Continuous Delivery for PE user.

You must have root or super user permissions to access the root console and perform this action.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Workspaces**.
3. Locate the workspace you wish to reassign and click **Reassign workspace**  .
4. Select the username of the new owner of the workspace, then click **Save changes**.

Ownership of the workspace is now transferred. The original owner of the workspace is automatically removed from the workspace. If necessary, the new owner can add the former owner as a workspace user by navigating to **Settings** > **Users**.

Managing access

Use the user access and management tools in Continuous Delivery for Puppet Enterprise (PE) to ensure that each user has the access and permissions appropriate to their role on the team, from read-only users to super users.

- Create a user account on page 73

After Continuous Delivery for Puppet Enterprise (PE) has been installed by an administrator on your team, each person who will use the software must create an individual user account.

Create a user account

After Continuous Delivery for Puppet Enterprise (PE) has been installed by an administrator on your team, each person who will use the software must create an individual user account.

> **Before you begin**
>
> Obtain the Continuous Delivery for PE web UI endpoint from the administrator on your team who installed the software.

1. Point your browser to the Continuous Delivery for PE web UI endpoint.
2. On the login page, click **Create an account**.
3. Fill in the registration form and create a username and password.
4. Click **Sign up**.

Congratulations! You now have a Continuous Delivery for PE account.

View your account credentials by clicking your username at the bottom of the navigation bar.

Adding users and creating groups

Continuous Delivery for Puppet Enterprise (PE) is built to facilitate collaboration between the members of your team. You can share resources by adding users to your workspace, and set up groups to manage the permissions granted to each member of your team.

Add a user to a workspace

Adding users to your workspace allows you to collaborate on projects and share account resources such as jobs and pipelines. You can add users in the **Settings** area of the Continuous Delivery for PE web UI.

> **Before you begin**
>
> Make sure each user you wish to add has signed up for a Continuous Delivery for PE account. You'll need their selected username or the email address they used when signing up.

1. In the Continuous Delivery for PE web UI, click **Settings** then click **Users**.
2. Click **+ Add users**.
3. On the **Add users** page, enter the username or email address of the user you wish to add to your workspace.
4. Check the box next to the name of each user you wish to add to the workspace, then click **Add users to workspace**.
5. Repeat these steps to add additional users.
6. When you're finished adding users, click **Done** to return to the **Users** screen, where you'll see the full list of users with access to your Continuous Delivery for PE workspace.

Create a new group and set group permissions

You can set up groups to organize users by permission level, job focus, geography, or other criteria. Permissions are the tasks members of a group can perform in Continuous Delivery for PE, such as adding new control repos, editing jobs, and deploying code changes. Assign permissions to a group based on that group's function and needs.

1. In the Continuous Delivery for PE web UI, click **Settings**.
2. In the **Groups** tab, click **+ Create new group**.
3. Enter a name and description for your new group.

> **Tip:** Use a name for your group that describes the group's function or permission level, such as "Read-only users" or "Module developers."

> **CAUTION:** Once submitted, group names and descriptions cannot be edited. If you wish to change the name or description of your group, you must delete the group and recreate it.

4. Select the permissions you wish to assign to this group. Group permissions are additive. If a user is a member of multiple groups, that user is able to perform all of the actions described by all of the permissions in all their assigned groups.

   You can assign permissions on only a certain subset of the modules in your workspace to a group. To define the module subset:

   a) In the modules permissions section, click  next to **Modules subset:**.
   b) Select the modules that you want to include in the subset. When the list is complete, click **Save selection**.
   c) Select which permissions you wish to give the user group. You can allow the group list, edit, and delete permissions on all modules in the workspace or only on your selected subset.

      If you create a module subset, but do not assign group any permissions on the subset, the subset will be lost when you save your permission selections.

   For an explanation of the scope of each permission, see the Permissions reference.
5. Click **Save and add users**.
6. On the **Add users** screen, select the workspace members who you wish to add to this group. Click **Add users to group**.
7. Click **Done**. Your user group is created, and you are redirected to the group page.

Review the group members and permissions on the group page, where you can also make any needed changes.

If you need to delete a group, first remove all members from the group and then click **Remove group**  .

Remove a user from a workspace

If you wish to remove a user from Continuous Delivery for PE workspace, you can do so in the **Settings** area of the Continuous Delivery for PE web UI.

> **Note:** If you remove a user from a Continuous Delivery for PE workspace, that user is automatically removed from all groups in that workspace.

1. In the Continuous Delivery for PE web UI, click **Settings** then click **Users**.
2. In the list of users, locate the user you wish to remove from your workspace.
3. Click **Remove user**  and confirm your action.
4. Repeat these steps to remove additional users from the workspace.

Permissions reference

The table below lists the group permissions available in Continuous Delivery for Puppet Enterprise (PE), along with a short explanation of each permission's scope.

Type	Permission	Definition
Control repos	Create	The ability to create new control repos in Continuous Delivery for PE.
Control repos	Delete	The ability to delete existing control repos.
Control repos	Edit	The ability to edit existing control repos. The ability to deploy code.
Control repos	List	The ability to view all existing control repos.
Modules	Create	The ability to create new module repos in Continuous Delivery for PE.
Modules	Delete	The ability to delete existing module repos.
Modules	Edit	The ability to edit existing module repos.
Modules	List	The ability to view all existing module repos.
Jobs	Create	The ability to create new jobs in Continuous Delivery for PE.
Jobs	Delete	The ability to delete existing jobs.
Jobs	Edit	The ability to edit existing jobs.
Jobs	List	The ability to view all existing jobs and job hardware capabilities.
Jobs	Run	The ability to run all existing jobs.
Users	Edit	The ability to add new users to Continuous Delivery for PE, and to remove existing users.
Groups	Create	The ability to create new user groups.
Groups	Delete	The ability to delete existing user groups.
Groups	Edit	The ability to edit existing user groups by adding or removing members and user permissions.
Groups	List	The ability to view all existing user groups and their permissions.
Integrations	Connect	The ability to create new source control or Puppet Enterprise integrations.

Type	Permission	Definition
Integrations	Disconnect	The ability to remove existing source control or Puppet Enterprise integrations.

Using the root console

Super users and the root user can access the root console in Continuous Delivery for Puppet Enterprise (PE). Use the root console to manage users' account credentials, change super user permissions, configure single sign-on, access all workspaces associated with the installation, transfer ownership of workspaces, and update your installation's settings.

To access the root console:

- If you are the **root user**, sign into Continuous Delivery for PE with the root account credentials established during installation.
- If you are a **super user**, select **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar.

To exit the root console, click the name of a workspace.

Designate super users

Any Continuous Delivery for PE user except the root user can be designated as a super user. Super users have access to the root console, where they can reset other users' credentials and delete users. Super users can also manage global shared job hardware and designate which users are able to manually approve deployments to protected Puppet environments.

You must have root or super user permissions to access the root console and perform this action.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Accounts**.
3. Locate the user's name in the list and select the **Super User** toggle, then confirm your action.

   The selected user is now a super user.

Change a user's password

Users can reset their own passwords at any time by clicking **Forgot your password?** on the Continuous Delivery for PE login screen. If you need to update the root user's password, change a user's password on their behalf, or revoke a user's access to Continuous Delivery for PE, use the root console.

You must have root or super user permissions to access the root console and perform this action.

CAUTION: Resetting a user's password does not automatically log the user out of Continuous Delivery for PE. If you reset a user's password in order to revoke their access to Continuous Delivery for PE, remember that the user can still access their account until their session expires or they log out.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Accounts**.
3. Locate the user whose password you wish to update by searching by username or email address. and click **User settings**  .
4. Click **Change password**. Enter and confirm the new password, then click **Change password**.

Reset a user's email address

Users can change their own email address at any time by clicking **Change email** on the **Profile** page. If you need to change a user's email address on their behalf, use the root console.

You must have root or super user permissions to access the root console and perform this action.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Accounts**.
3. Locate the user whose password you wish to update by searching by username or email address. and click **User settings** .
4. Enter the new email address for the user and click **Change email**.

Delete a user

Use the root console to permanently delete a user from Continuous Delivery for PE. Delete users with caution: deleting a user also deletes all artifacts that user has created in Continuous Delivery for PE, including workspaces, pipelines, jobs, integrations, control repos, and module repos.

You must have root or super user permissions to access the root console and perform this action.

1. Log into the root console by selecting **Root console** from the workspaces menu at the top of the Continuous Delivery for PE navigation bar or signing in as the root user.
2. Click **Accounts**.
3. Locate the user's name in the list and click **Delete user** .
4. In the confirmation pane, confirm that you want to permanently delete the user by entering the user name in the box and clicking **Yes, delete user**.

The user and all artifacts associated with that user are permanently deleted from your Continuous Delivery for PE installation.

Reviewing node inventory

The nodes from all Puppet Enterprise (PE) instances integrated with a workspace are shown on that workspace's **Nodes** page. This page gives you a federated view of all the nodes that your workspace deploys code to.

Enabling the Nodes page

In order to display node data, the **Nodes** page query service listens on port 8888. To make this port available to your Continuous Delivery for PE instance, upgrade to version 2.0.1 or later of the `puppetlabs-cd4pe` module.

If the port is not accessible (generally because the module has not yet been upgraded), you'll see an `Unable to connect to the query service on http://<cd4pehost>:8888>/query` message in the web UI.

Using the Nodes page with SSL enabled

If you have enabled SSL for your Continuous Delivery for PE installation, you cannot view the **Nodes** page using an HTTP connection. Connect using HTTPS to reach the page.

Firefox users who have enabled SSL and are using a self-signed certificate must add the certificate to the Firefox trust store.

Customize your node table

The **Nodes** page shows a list of every node from every PE instance integrated with your workspace, along with basic information about each node. You can customize your page view to display node facts that are relevant to your work.

By default, the node table displays the following information about each node:

- Node name

> **Note:** Click the node name to view all facts for the node and a list of all available PE reports for the node.

- Time since the node's most recent PE report was generated
- Node run status for the most recent Puppet run
- The PE server the node is associated with
- The value of the `ipaddress` fact
- The value of the `operatingsystem` fact

You can add additional columns showing other fact values to the node table, or remove default columns that do not meet your needs.

1. In the Continuous Delivery for PE web UI, click **Nodes**.
2. Click **Columns +/-** to open the column selector.
3. To add a new column, search for the fact value you wish to view and click **Add**. The fact value appears in the list of columns.
4. To determine which columns are shown or hidden, use the checkmarks next to the list of columns to indicate which columns you want to display.
5. When your column list is ready, click **Apply**. Your changes are reflected in the table.

Testing Puppet code with jobs

Jobs are fully customizable tests for your Puppet code. You can create a job that runs any sort of test you wish, from module validation to linting.

> **Important:** Make sure you have set up job hardware and installed any needed software (such as Docker or Puppet Development Kit) before you attempt add one of these jobs to a pipeline.

What is a job?

In Continuous Delivery for PE, jobs are tests for your Puppet code. Jobs are created and stored in the web UI, where you can run them on demand and add them to an automated pipeline that runs tests every time new code is committed to a repository.

Jobs are fully customizable and can be written to test any aspect of your code. Jobs can be written to take advantage of Puppet Development Kit (PDK) and other testing tools created and maintained by the Puppet community, such as:

- rspec-puppet
- onceover
- puppet-lint
- rspec-puppet-facts

It's important to be aware of any prerequisites or dependencies needed to run your jobs, and to ensure that your job hardware has the necessary software, operating systems, and other resources installed before attempting to run the job.

Pre-built job reference

Continuous Delivery for PE offers users six pre-built Docker-based jobs for testing control repos and modules. These jobs run inside a Docker container, and must be run on job hardware on which you've set a Docker capability.

Validate the syntax of a control repo's Puppetfile

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	control-repo-puppetfile-syntax-validate
Description	Validate that a control repo's Puppetfile is syntactically correct
Commands: Job	`rake -f /Rakefile r10k:syntax`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Validate the syntax of a control repo's Puppet templates

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	control-repo-template-syntax-validate
Description	Validate that a control repo's Puppet templates are syntactically correct
Commands: Job	`rake -f /Rakefile syntax:templates`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Validate the syntax of a control repo's Hiera data

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	control-repo-hiera-syntax-validate
Description	Validate that a control repo's Hiera data is syntactically correct
Commands: Job	`rake -f /Rakefile syntax:hiera`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Validate the syntax of a control repo's Puppet manifest code

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	control-repo-manifest-validate
Description	Validate that a control repo's Puppet manifest code is syntactically correct
Commands: Job	`rake -f /Rakefile syntax:manifests`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Validate the syntax of a module's Puppet manifest code

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	module-pdk-validate
Description	Validate that a module's Puppet manifest code is syntactically correct
Commands: Job	`pdk validate --parallel`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Run rspec-puppet unit tests on a module

To add this job to your Continuous Delivery for PE instance, click **New job**. Fill in the fields indicated with the information provided, then click **Create job**.

Field	Content
Job Name	module-rspec-puppet
Description	Run rspec-puppet unit tests on a module
Commands: Job	`pdk test unit`
Docker Configuration	Run this job in a Docker container
Docker Image Name	puppet/puppet-dev-tools
Job Hardware: Capabilities	DOCKER

Before running this job, configure job hardware and ensure that Docker is installed.

Sample non-Docker-based module jobs

This section lists sample jobs that you can use with Continuous Delivery for PE. These jobs can be entered as-is into the **New Job** creation screen in the Continuous Delivery for PE web UI, or you can make alterations to suit your deployment's needs.

> **Note:** When composing a non-Docker-based job, the `$REPO_DIR` environment variable can be used to reference the directory that houses the relevant module repo.

Puppet Development Kit validation tests

Use the sample jobs below to validate your module code against PDK. Select the version appropriate to your operating system.

In order to use this job successfully, you must:

- Install PDK on your job hardware
- Install `puppet-agent` on your job hardware

Job name	Description	Commands	Capabilities
module-pdk-validate-linux	Validate via PDK	`pdk validate`	LINUX
module-pdk-validate-windows	Validate via PDK	`powershell.exe -c "pdk validate"`	WINDOWS

Puppet Development Kit `rspec-puppet` tests

Use the sample jobs below to run unit tests on your module code with `rspec-puppet`. Select the version appropriate to your operating system.

In order to use this job successfully, you must:

- Install PDK on your job hardware
- Install `puppet-agent` on your job hardware

Job name	Description	Commands	Capabilities
module-pdk-test-unit-linux	Run unit tests via PDK	`pdk test unit`	LINUX
module-pdk-test-unit-windows	Run unit tests via PDK	`powershell.exe -c "pdk test unit"`	WINDOWS

Sample non-Docker-based control repo jobs

This section lists sample jobs that you can use with Continuous Delivery for PE. These jobs can be entered as-is into the **New Job** creation screen in the Continuous Delivery for PE web UI, or you can make alterations to suit your deployment's needs.

> **Note:** When composing a non-Docker-based job, the `$REPO_DIR` environment variable can be used to reference the directory that houses the relevant control repo.

Syntax validation

This job validates the syntax of everything in your control repo.

In order to use this job successfully, you must:

- Use a *nix host

- Install `puppet-agent` on your job hardware

Job name	Description	Commands	Capabilities
control-repo-validate-linux	Validate syntax	[See below]	LINUX

```
#!/bin/bash

shopt -s globstar nullglob
green="$(tput setaf 2)"
red="$(tput setaf 1)"
reset="$(tput sgr0)"

for f in **/**pp; do
   [[ $f =~ plans/ ]] && continue

   if puppet parser validate "$f"; then
      echo "${green}SUCCESS: $f${reset}"
   else
      echo "${red}FAILED: $f${reset}"
      failures+=("$f")
   fi
done

if (( ${#failures[@]} > 0 )); then
   echo "${red}Syntax validation on the Control Repo has failed in the
 following manifests:"
   echo -e "\t ${failures[@]}${reset}"
   exit 1
else
   echo "${green}Syntax validation on the Control Repo has succeeded.
${reset}"
fi
```

Puppet linter

This job checks the Puppet code in your control repo for programming and stylistic errors.

In order to use this job successfully, you must:

- Use a *nix host
- Install `puppet-agent` on your job hardware

Job name	Description	Commands	Capabilities
control-repo-lint-linux	Lint Puppet code	[See below]	LINUX

```
#!/bin/bash

shopt -s globstar nullglob
green="$(tput setaf 2)"
red="$(tput setaf 1)"
reset="$(tput sgr0)"

sudo /opt/puppetlabs/puppet/bin/gem install puppet-lint || {
   echo "${red}Failed to install puppet-lint gem"
   exit 2
}

LINT_OPTS=("--fail-on-warnings" "--no-documentation-check"
 "--no-140chars-check" "--no-autoloader_layout-check" "--no-
class_inherits_from_params_class-check")
```

```
for f in **/**pp; do
   [[ $f =~ plans/ ]] && continue

   if /opt/puppetlabs/puppet/bin/puppet-lint "${LINT_OPTS[@]}" "$f"; then
      echo "${green}SUCCESS: $f${reset}"
   else
      echo "${red}FAILED: $f${reset}"
      failures+=("$f")
   fi
done

if (( ${#failures[@]} > 0 )); then
   echo "${red}Puppet-lint validation on the Control Repo has failed in the
 following manifests:"
   echo -e "\t ${failures[@]}${reset}"
   exit 1
else
   echo "${green}Puppet-lint validation on the Control Repo has succeeded.
${reset}"
fi
```

Key concepts

Working with Git branches in Continuous Delivery for PE

Continuous Delivery for PE is designed to watch for Puppet code changes you make in Git and help you deploy those changes to your environment node groups. You and Continuous Delivery for PE work together, so it's important to understand what to do in Git, and what Git work Continuous Delivery for PE handles for you.

Continuous Delivery for PE automates management of Git branches for Puppet environments, letting you focus on Git workflows for developing and deploying changes to your code.

You and your team own the master branch in Git, as well as any other development-focused branches you create, such as feature, hotfix, or release branches. In simple Git workflows, the master branch is designated as the branch to file pull requests against, tag releases off of, and treat as the leading edge of new development.

It is also your responsibility to create new environment branches when you first set up environment node groups. Environment branches are long-lived Git branches used to control which version of code is made available in corresponding Puppet environments. Continuous Delivery for PE owns environment branches after they're created, deploying changes to them and keeping them up to date. You and your team won't make changes to these branches

directly, because manual changes to environment branches get overwritten whenever an automated deployment occurs.

While you focus on making commits to master and getting pull requests merged, Continuous Delivery for PE works in your environment branches on your behalf, making sure your commits or changes are tested, deployed, and promoted to your environment node groups in accordance with the control repo pipelines you create and your manual deployment directives.

Continuous Delivery for PE also uses a small set of Git branches for bookkeeping, which are not part of the standard Git process. These branches are visible to users, but should be protected so that users cannot modify them. Continuous Delivery for PE manages these bookkeeping branches automatically.

Organizing Puppet content in your source control repository

Most of the Puppet content that developers interact with is infrastructure-as-code, which is versioned and committed to a Git source-control repository.

When using Continuous Delivery for PE, every Git repository, regardless of its type or contents (control repo or module), works the same way and has the same key components:

- A `master` branch that tracks the latest developing code
- Tags to identify meaningful code versions
- Short-lived "feature" branches that are created and deleted as part of the code development lifecycle

> **Tip:** Most "Git 101" tutorials describe how to interact with standard Git repositories of this type, and provide an overview of the basic branching and merging process.

Control repos

A control repo is a hub for Puppet configuration content. In its purest form, that's *all* it is. More typically, however, the control repo serves as a hub for modularized content, and also contains some directly embedded content. Typically, the embedded content includes the "role" module, the "profile" module, and Hiera data. (In a pure control repo, this content is modularized rather than embedded.)

The control repo contains a **Puppetfile**. The Puppetfile is the switchboard or ledger that makes the control repo a hub. It is a file that enumerates all of the modular content Puppet will incorporate and make available. Reading the Puppetfile should convey a good sense of all the Puppet content being used at a given site or Puppet implementation.

Depending on how much content has been embedded in the control repo, it might also contain a small handful of Puppet configuration files, and possibly some directories for the embedded content.

Puppet module repos

A Puppet module is a collection of content laid out in a particular way. A module can contain desired state code, Bolt tasks or plans, Puppet extensions, or a variety of other things. The module is the standard content container for everything Puppet.

Puppet knows how to consume modules automatically, provided they are placed somewhere in its `modulepath`.

Other "non-module" repos

It's sometimes useful to partition off certain types of Puppet for development purposes, even if the content is not technically a Puppet module. The prime example of this is Hiera data. Hiera data can be kept in a dedicated Git repository and evoked by the control repo using the Puppetfile, just like any standard Puppet content.

Other use cases for carving off non-module Git repositories exist, but they are rare. The commonality is that all such content will be committed to a Git repository and be referenced in the control repo's Puppetfile.

Puppet typically doesn't consume non-module content automatically, and correct consumption of this content requires some kind of configuration in one of the control repo's configuration files.

Understanding the Continuous Delivery for PE workflow

Continuous Delivery for Puppet Enterprise (PE) enables you to deliver change to your organization's infrastructure-as-code. Use the workflow outlined here to develop and deploy changes with Continuous Delivery for Puppet Enterprise (PE).

> **Tip:** Before you begin, review Working with Git branches in Continuous Delivery for PE.

Workflow phase 1: Developing changes

The high-level process for developing a change is always the same regardless of whether the content is in the control repo, a module repo, or a non-module repo. The change development process is driven through Git.

Here's the high-level process for developing a change:

1. Using Git, create a feature branch from the `master` branch. This feature branch is where you'll develop your change.
2. Work on your change. Edit files, run tests, and do anything else necessary in order to realize, in code, the change you want to make. The details of what files you edit and how you test your change locally will vary depending on which kind of content you're working on.
3. Using Git, commit all the files you've changed to your feature branch. You may be committing frequently as you iterate, or you may not commit until you're fully satisfied with your work. How much you like to involve (or not involve) Git in your process up to this point is totally up to you.
4. Submit a pull/merge request to have your feature branch merged into `master`.
5. Assuming whatever CI or process requirements your organization uses are satisfied, you or someone else merges your feature branch into `master`. Once the merge is complete, your change is now available and ready to be put on the path to production, and you can safely delete your feature branch.

At this point your change is finished, as it has been merged into the `master` branch. Note that a finished change is *not* the same as a deployed change; a finished change is a change ready to be tested and deployed using the deployment process described below.

A note on continuous integration

When you submit a pull/merge request against a Puppet content repo (a control repo, a module repo, or a non-module repo), the continuous integration (CI) pipeline configured in Continuous Delivery for PE for that content repo runs and reports back success/failure results in your pull/merge request.

Pipelines in Continuous Delivery for PE are not cleanly separated into CI and CD. Instead, you see a single contiguous pipeline. However, it is a fair approximation to say that everything in a Continuous Delivery for PE pipeline that occurs before a special marker called the *Pull Request Gate* is the CI portion of the pipeline, and is (usually) focused on validating that proposed changes pass requisite acceptance tests or governance requirements.

The CI portion of a Continuous Delivery for PE pipeline therefore aids the development workflow.

Workflow phase 2: Deploying changes

Once a proposed change has been merged into the `master` branch of a Puppet content repo, the new content version (in the `master` branch) containing the change needs to be deployed.

In broad terms, to deploy a change means to select a version of content, select a target environment, and cause the selected change to take effect in the selected environment.

In Continuous Delivery for PE, this process looks like:

1. Pick a version.
2. Pick a target environment.
3. Optionally, set some deployment parameters.
4. Click "Deploy."

A version is any commit in the history of the `master` branch. There are no chronological requirements for picking a version to deploy. Deploying a version will neither change the contents of the `master` branch nor affect it in any way.

Environments

An environment is a target set of nodes to which content versions can be deployed. Environments are configured using Environment Node Groups in the PE console.

> **Note:** See Create environment node groups for more information.

Per the definition of deploying above, environment node groups need to be configured as a prerequisite to deploying any content versions. At a minimum, the environment you want to target must be defined as an environment node group.

Some basic characteristics of defining environment node groups:

- Each node can only belong to a single environment node group.
- The environment node group can be named anything that makes sense to your organization.
- The environment node group must be assigned a "Puppet environment."
  - The functional importance of a "Puppet environment" relates to how Continuous Delivery for PE performs bookkeeping of the environment's content in Git (using the bookkeeping branches mentioned earlier).
  - Each environment node group should be assigned a unique "Puppet environment" value. This ensures simple bookkeeping and that change deployments do not affect nodes they are not supposed to.

Deploying changes from control repos

The basic process:

1. Pick a version.
2. Pick a target environment.
3. Optionally, set some deployment parameters.
4. Click "Deploy."

Because a control repo contains a Puppetfile, which is the switchboard or ledger to which all modularized content is attached, how that modularized content is deployed in a target environment depends on the Puppetfile deployed to that environment.

Modularized content can be attached to a Puppetfile *statically*, such that the modularized content cannot be independently deployed or updated, or the content can be attached *dynamically*, so that the modularized content can be deployed independent of the control repo using a different Continuous Delivery for PE pipeline or deployment action.

If modularized content is defined statically in the Puppetfile, the only way to update it is to develop a change to the control repo itself, which modifies the Puppetfile to update the static module, and then deploy that change to the control repo.

Deploying changes from module repos

The basic process:

1. Pick a version.
2. Pick a target environment.
3. Optionally, set some deployment parameters.
4. Click "Deploy."

The inverse of the note about control repos above is that to be able to deploy module changes this way, the control repo version deployed to the target environment must have a Puppetfile that attaches this modularized content

dynamically. If the target environment does not have such a Puppetfile, it is not possible to independently deploy the module changes.

Pipelines, or the path to production

Many organizations have a semi-linear sequence of target environments to which a change will be deployed—a sequence which terminates at the highest-value target environment. This final target is frequently called "production."

A pipeline in Continuous Delivery for PE is automation reflecting this sequence of deployments. It is a progression of deployments (and other actions) which, when triggered, will run in a defined manner, pausing, terminating, or continuing after each step as configured to by the pipeline creator.

The pipeline can contain many kinds of actions besides deployments. As mentioned in the "note on continuous integration" section above, the first part of a Continuous Delivery for PE pipeline up until the special *Pull Request Gate* step is typically CI validation and testing actions, rather than deployment actions. Deployment actions will usually not be placed in the pipeline until after this step.

The *continuous* element of a pipeline is in reference to what triggers it, or when it is run. Pipelines typically start automatically as soon as a new change has been merged to the `master` branch of a content repository.

Deployment pipelines typically deploy validated changes automatically up to some level lower than "production," but pause or wait for human approval before continuing to run and deploy to more sensitive, higher-tiered environments.

Using the feature branch workflow

Use the feature branch workflow to safely and efficiently move new code from an isolated development environment through initial testing and validation and then into your organization's staging, review, and promotion to production process. This guide provides an outline of this core Continuous Delivery for PE workflow; adapt the steps as needed to meet your team's requirements and best practices.

Part 1: Develop and test code changes

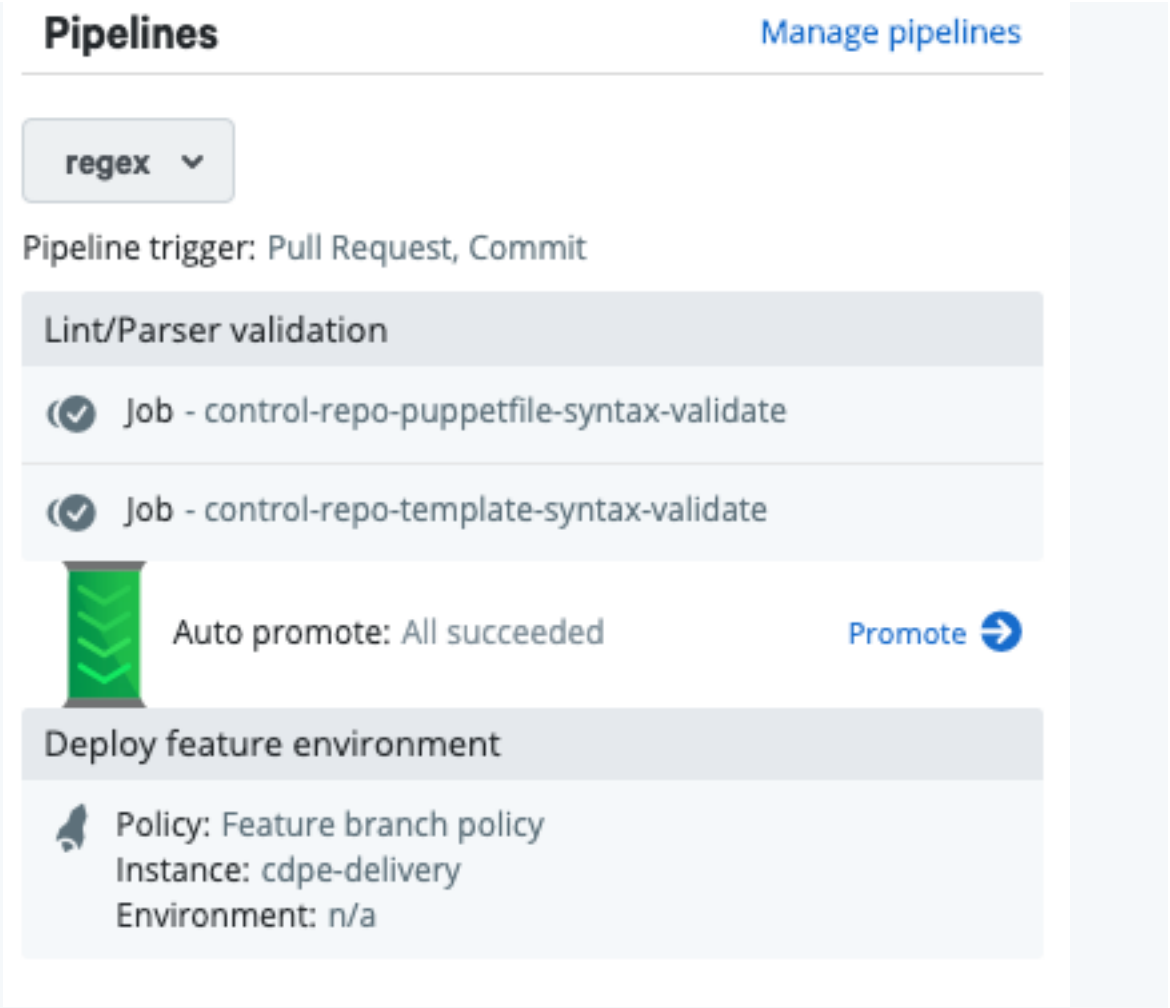
The first phase of the feature branch workflow focuses on writing new code in a feature branch, then testing and validating that code.

Before you begin

Create a regex branch pipeline for the control repo or module repo you're working on. This pipeline can be as simple or complex as you need it to be, but at minimum it must:

- Use commits as a pipeline trigger
- Contain a deployment using the feature branch deployment policy

We also strongly advise including jobs that perform syntax validation tests in your regex branch pipeline before the deployment stage. Here's a sample regex branch pipeline as it appears in the web UI:



Here's what the pipeline above looks like when expressed as code:

```
pipelines:
  /feature_.*/:
    triggers:
    - "PULL_REQUEST"
    - "COMMIT"
    stages:
    - name: "Lint/Parser validation"
      steps:
      - type: "JOB"
        name: "control-repo-puppetfile-syntax-validate"
        concurrent_compilations: 0
        all_deployments: false
      - type: "JOB"
        name: "control-repo-template-syntax-validate"
        concurrent_compilations: 0
        all_deployments: false
      auto_promote: "all_succeeded"
    - name: "Deploy feature environment"
      steps:
      - type: "DEPLOYMENT"
        name: "Feature branch deployment on cdpe-delivery"
        policy:
          name: "cd4pe_deployments::feature_branch"
        concurrent_compilations: 0
```

```
        all_deployments: false
        pe_server: "cdpe-delivery"
      auto_promote: false
```

1. In your source control system, navigate to the control repo or module repo you want to update.
2. Create a feature branch. Make a new branch based on the master branch and name it `feature_<BRANCHNAME>`, substituting a relevant ticket number, fix description, or feature name for `<BRANCHNAME>`.

   Continuous Delivery for PE automatically recognizes feature branches by their names. By default, the software uses `feature_.*` as the regular expression that triggers regex pipelines, so it's important to give your feature branch the `feature_` prefix.

   > **Tip:** If you need to use a different naming convention for your control repo or module repo's feature branches, you can change the regular expression in the web UI by clicking **Manage pipelines**.

3. On your feature branch, make your code changes.
4. When your work is complete, commit the changes to the feature branch and push the commit to your source control system. This commit triggers your regex branch pipeline. You can monitor the pipeline's progress in the web UI.

   When your pipeline is triggered, Continuous Delivery for PE runs any jobs and impact analysis tasks you've set up. If the promotion conditions set in your pipeline are met (if all the jobs succeed, for example), the pipeline starts a feature branch deployment.

   How does a feature branch deployment work? Using Code Manager, Continuous Delivery for PE deploys the code in your commit to a Puppet environment with the same name as the branch that triggered the pipeline. If an environment with this name doesn't already exist (as is most likely the case), Continuous Delivery for PE creates it.
5. When the feature branch deployment is complete, run Puppet on a test node, specifying the special environment created by Continuous Delivery for PE. This Puppet run will apply your changes to the test node so you can review them. You can run Puppet from the command line or from the PE console:
   a) **From the command line:** Run `puppet job run --nodes <TEST_NODE_NAME> --environment <feature_BRANCHNAME>`

      > **Tip:** For more on the `puppet job run` command, see Running Puppet on demand from the CLI.

   b) **From the PE console:** Navigate to the **Run Puppet** page and create a job with the following settings:

      - Job description: Enter a description for your test run
      - Environment: Select **Select an environment for nodes to run in** and choose the environment that matches your feature branch name
      - Schedule: Now
      - Run Mode: Leave these options unchecked
      - Inventory: Select **Node list** and add the name of your test node

      Click **Run job**.
6. When the Puppet run is complete, navigate to your test node and review your changes. If further work is needed, repeat steps 3 through 5 until your code is ready for deployment to production.

   When you're satisfied with your new code, move on to **Part 2: Review and merge to production**.

Part 2: Review and merge to production

In the second phase of the feature branch workflow, the new code is reviewed and merged to the master branch, then deployed to production. Along the way, Continuous Delivery for PE provides checks and safeguards to ensure new code is only sent to production nodes when it has been fully vetted.

Before you begin

Make sure your master branch pipeline uses pull requests as a pipeline trigger and includes a PR gate.

1. In your source control system, create a pull request from your `feature_<BRANCHNAME>` branch to the master branch.

   > **Note:** If you're a GitLab user, you might be more familiar with the term "merge request" than "pull request." Although the two terms refer to the same concept and can be used interchangeably, for the sake of simplicity and consistency, the Continuous Delivery for PE web UI and these instructions use the term "pull request" (PR).

2. The pull request triggers a run of your master branch pipeline, which tests your PR code against the jobs and impact analysis report tasks included in the pipeline before the PR gate. You can monitor the pipeline's progress in the Continuous Delivery for PE web UI.

   The success or failure of each stage of the pipeline is also displayed on the pull request's page in your source control system.

3. The appropriate stakeholders on your team can now review the results of the jobs and impact analysis reports generated by the pipeline. If the PR is approved, merge it into the master branch.

   > **Note:** After the PR is merged, you can safely delete the `feature_<BRANCHNAME>` branch you've been using from your source control system.

   Merging the PR (which your version control system views as a commit to the master branch) automatically re-triggers the master branch pipeline. The pipeline starts over from the top, re-running all the jobs and impact analysis tasks and making sure the newly updated master branch code is fully tested and vetted.

   Since this pipeline run was triggered by a commit, the pipeline proceeds past the PR gate and performs any tasks included in the pipeline after the PR gate, such as additional jobs or deployment to a staging environment.

4. Once the code is deployed to a staging environment, perform additional testing by running Puppet against test or staging nodes, specifying the staging environment. The scope of this testing should be determined by your team or company best practices.

5. When testing is complete and the code changes have been approved, promote the code to the production environment.

   In most cases, the master branch pipeline does not auto-promote to a deployment to the production environment. A stakeholder must click **Promote** in the pipeline to trigger the final stage. This kicks off a deployment to the production environment.

   However, if the production environment is a protected environment, the deployment requires review and approval from a member of a designated approval group before it can proceed. The deployment remains in a pending state until an approval decision is provided.

   > **Note:** See Require approval for deployments to protected Puppet environments for more information on setting up approval groups.

6. Finally, if necessary, run Puppet to apply the new code to the nodes in your production environment. Depending on which deployment policy you've used, a Puppet run might be included in the deployment Continuous Delivery for PE performs. Otherwise, you can trigger Puppet manually, or wait for the next scheduled Puppet run.

When the Puppet run is complete, your code changes are officially deployed to production.

Constructing pipelines

Pipelines enable the "continuous" in Continuous Delivery for Puppet Enterprise (PE). Constructing a pipeline means defining the work that needs to happen to make sure every new line of Puppet code is ready for deployment. Once your pipeline is set up, this work happens automatically each time the pipeline is triggered.

Stages and tasks

Pipelines in Continuous Delivery for PE are made up of stages and tasks. Tasks include deployments, impact analyses, and jobs to test code; stages group tasks into a series of sequential phases.

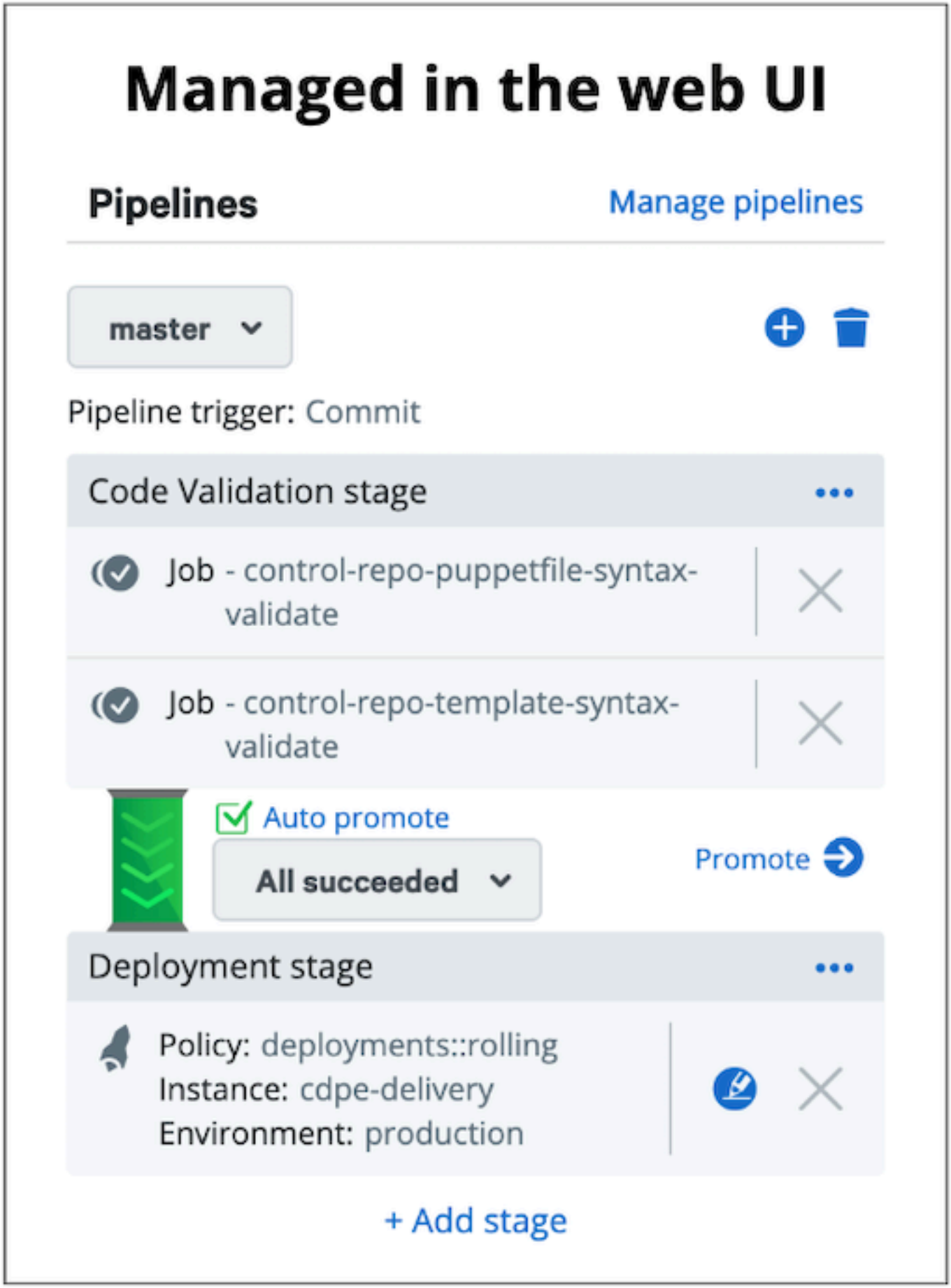
If you want to set up some logic of the "if this job succeeds, then run the next one, otherwise stop and tell me what happened," variety, use stages to break up the pipeline into a series of incremental steps. You can set your pipeline to require manual promotion before moving on to the next stage, or to promote automatically when certain conditions are met.

Building and managing pipelines: web UI or code?

You have two options when deciding how to build and manage your pipelines: using the controls in the web UI, or defining pipelines in a YAML file.

- Building a pipeline with the web UI is the simplest method for building and managing pipelines. The web UI controls allow you to more easily make iterative changes.
- Building a pipeline as code is more complex, but ideal if you need a record of changes to your pipeline over time, or if you want to avoid hand-creating similar pipelines for many control repos or modules. Using the pipelines-as-code method also lets you commit changes to your pipeline alongside changes to your Puppet code that will necessitate a new pipeline definition.

> **Important:** You can't use a mixture of the two methods within a single control repo or module. User controls are disabled for pipelines that are managed as code.



Constructing pipelines in the web UI

Build and manage pipelines for your control repo or module in the Continuous Delivery for Puppet Enterprise (PE) web UI using the tools built into the interface. If you complete the tasks on this page in the order presented, you'll set up and learn to use a basic pipeline.

Create a pipeline

Set up a pipeline to enable automatic testing of newly added code by adding stages and jobs in the Continuous Delivery for PE web UI.

1. In the Continuous Delivery for PE web UI, click **Control repos**. Click the name of the control repo you wish to set up a pipeline for.

   Continuous Delivery for PE automatically creates a pipeline for the branch you selected when setting up the control repo (the master branch). You'll see this branch name at the top of the **Pipelines** area of the web UI.

> **Tip:** You can also set up pipelines for other branches in this repo, in order to automatically run tests on the code changes that Continuous Delivery for PE makes on your behalf. To create a pipeline for a different branch, click **Add pipeline**  . You can undo this action with **Delete pipeline**  .

2. Every pipeline needs at least one stage. Click **+ Add stage** and select **Jobs**.
3. Select a job from the list, and click **Add stage**. Your job is added to the pipeline. Click **Done**.

   If you need to run a single test on your new code, your pipeline can be this simple. But if you wish to add additional tests, you can either add them to the existing stage by clicking **Add job**, or create another stage.
4. Create a second stage in your pipeline by clicking **Add another stage** and selecting **Job**. Select a job from the list and click **Add stage**, then **Done**.

> **Tip:** If you wish to remove or reorder the stages in your pipeline, click **More actions**  . To delete all stages in a pipeline and start fresh, click **Delete pipeline**  .

5. You now have a two-stage pipeline with a promotion step between the stages. Select **Auto promote** and choose a promotion condition from the options.

   The promotion condition you select tells the pipeline what to do at the end of the pipeline stage. The available permission conditions, listed from most to least restrictive, are:

   - **All succeeded**: The pipeline moves on to the next stage only if all the steps in the current stage finish with a "succeeded" or "done" status.
   - **All completed**: The pipeline moves on to the next stage if all the steps in the current stage finish with a "succeeded," "done," or "failed" status. The pipeline does not move on if any step in the current stage is canceled.
   - **Any succeeded**: The pipeline moves on to the next stage if at least one of the steps in the current stage finishes with a "succeeded" or "done" status.
   - **Any completed**: The pipeline moves on to the next stage if at least one of the steps in the current stage finishes with a "succeeded," "done," or "failed" status. The pipeline does not move on if all steps in the current stage are canceled.

Your pipeline is ready for use.

Create a regex branch pipeline

Most pipelines are associated with a single branch in your control repo or module repo. A regex branch pipeline is configured to recognize and act on changes to any branch on your control repo or module repo that has a name matching the regular expression you set.

A regex branch pipeline lets you use a single pipeline for all the feature branches you and your team create and destroy in the process of developing new code. When you create a commit or pull request to a feature branch whose name uses the naming convention you've established for your regex branch pipeline, Continuous Delivery for PE automatically runs the regex pipeline.

1. In the Continuous Delivery for PE web UI, navigate to a control repo or module repo.
2. Click **Add pipeline**  .
3. In the **Add pipeline** window, select **Branch regex**.

   Only one regex branch pipeline can be created for each module repo or control repo. If you don't see the option to select **Branch regex** at the top of the window, a regex branch pipeline already exists for this repo.
4. Enter the regular expression that your regex branch pipeline will use to recognize feature branches that should trigger pipeline runs. The default regular expression is `feature_.*`. If you use the default regex, your feature branches must use a `feature_<BRANCHNAME>` naming convention.

   > **Tip:** All members of your team must use this naming convention when creating feature branches in order for their changes to trigger the regex branch pipeline. We strongly recommend using the default regular expression or one that's similarly simple and memorable.
5. Click **Add pipeline**. The regex branch pipeline is created.
6. **Optional.** To adjust the settings for your regex branch pipeline, click **Manage pipelines**. On this screen, you can set whether your regex branch pipeline will be triggered by commits, pull requests, or both.
7. Build out your pipeline by adding jobs, impact analysis tasks, and deployments.

You now have a pipeline configured to run in response to trigger conditions on any branch named with the naming convention you've selected.

Test code automatically with a pipeline

Every time you commit new code to your development or master branch, the pipeline runs the jobs you've set up to test the code and reports any errors or failures.

1. To see the pipeline in action, make a trivial change (such as adding a new line to the README file) in the master branch of your control repo. Commit your change.

2. In the Continuous Delivery for PE web UI, click **Control repos**, then click the name of the control repo you just committed code to.
3. The **Events** area now shows you the push event initiated by your code commit, and an entry for each job in your pipeline. The jobs report their status here, and update to show their success or failure when the job run is complete.

   Each event summary includes a link to the commit in the control repo. Job events also include a job number; clicking on this number takes you to the **Job details** page, where you can see the job's log for its run and review error messages.

Test pull requests automatically

When "pull request" is enabled as a pipeline trigger, your pipeline automatically tests new code each time a pull request is opened against the relevant branch. By setting up a PR gate in the pipeline, you can ensure that the relevant tests run on the new code, but stop the pipeline before it proceeds to further tests or a deployment.

> **Note:** If you're a GitLab user, you might be more familiar with the term "merge request" than "pull request." Although the two terms refer to the same concept and can be used interchangeably, for the sake of simplicity and consistency, the Continuous Delivery for PE web UI and these instructions use the term "pull request" (PR).

1. Add a pull request (PR) gate to your pipeline. A PR gate indicates that any pipeline runs triggered by a pull request stops at a certain point in the pipeline. This allows you to automatically run acceptance tests on new pull requests, but to manage additional testing or deployment of the new code manually based on the results of the initial automated tests.

   a) In your pipeline's first stage, click **More actions**  . Click **Add item to stage**, and select **Pull request gate**.

   b) Click **Add PR gate**. When the gate has been added, you'll see a success message. Click **Done**.

   > **Important:** You can add only one PR gate to a pipeline. If you decide to move the placement of a PR gate, delete the existing gate and create a new one in the desired stage.

2. To see the PR gate in action, switch to a branch in your control repo other than master. Make a trivial change (such as adding a new line to the README file) in your control repo, and commit your change, then open a pull request against the master branch for this commit.

   > **Remember:** The pull request must be made against the branch the pipeline is set up to use in order to trigger the pipeline.

3. In the Continuous Delivery for PE web UI, click **Control repos**, then click the name of the control repo you just committed code to.
4. The **Events** area now shows you the pull request event initiated by your code commit, and an entry for the job in your pipeline that precedes the PR gate. Note that the job in the pipeline's second stage, beyond the PR gate, has not been triggered.

Deploy code automatically with a pipeline

You can use a control repo pipeline to automatically deploy new code to a specified set of nodes every time a commit is made.

1. In the Continuous Delivery for PE web UI, click **Control repos**. Click the name of the control repo you've created a pipeline for.
2. Click **+ Add stage**.
3. In the **Add new stage** window, select **Deployment**.
4. Select your Puppet Enterprise instance and the node group you wish to deploy changes to every time this pipeline runs.

5. Select a deployment policy. For purposes of this getting started guide, select **Direct deployment policy**.

   See Deployment policies to learn more about the four deployment policies and how they work.
6. Optional: Set termination conditions for this pipeline's deployments, and choose the number of nodes that can fail before the deployment is stopped.
7. Click **Add Stage** and close the **Add new stage** window. Your new stage, showing the details of the deployment, is added to the pipeline.
8. To see the pipeline in action, make a trivial change (such as adding a new line to the README file) on the master branch of your control repo. Commit your change.

   The control repo **Events** area now shows you the push event initiated by your code commit, and a deployment event. The deployment reports its status here, and updates to show its success or failure when the deployment is complete.

   Each event summary includes a link to the commit in the control repo. Deployment events also include a deployment number; clicking on this number takes you to the deployment's details page, where you can see more information.
9. You can run the same deployment again from the web UI by retriggering the webhook. In the **Events** area, locate the push event you wish to rerun, and click **View webhook**.

   The webhook's request and response data is shown in the **Webhook data** pane, which can be useful for troubleshooting.
10. Click **Redeliver webhook**, and confirm your action. The **Events** area now shows the new event triggered by the redelivery of the webhook.

> **Note:** If you've made changes to your pipeline since the last time this webhook was delivered, the redelivered webhook follows the current pipeline's sequence and rules.

Constructing pipelines from code

Managing your pipelines with code, rather than in the web UI, lets you maintain a record of pipeline changes over time. When you elect to manage pipelines with code, a `.cd4pe.yaml` file with the pipelines' definitions for a control repo or module is stored in your source control system alongside the Puppet code for that control repository or module.

Configuring your pipelines for management with code

To begin managing your existing pipelines with code, or to create new pipelines using a `.cd4pe.yaml` file, follow the instructions on this page appropriate to your circumstances.

Have you added your control repo or module to Continuous Delivery for PE?	Do you have a `.cd4pe.yaml` file containing the pipeline definitions you want to use?	Follow these instructions
yes	not yet	Convert your existing pipelines to code
yes	yes	Update your pipelines using a new `.cd4pe.yaml` file

Have you added your control repo or module to Continuous Delivery for PE?	Do you have a `.cd4pe.yaml` file containing the pipeline definitions you want to use?	Follow these instructions
not yet	yes	Create new pipelines using a `.cd4pe.yaml` file

Convert your existing pipelines to code

If you've created pipelines using the Continuous Delivery for PE web UI, but now want to manage those pipelines with code, Continuous Delivery for PE will render your existing pipelines in YAML format.

Before you begin

Add a control repo or module to Continuous Delivery for PE and set up a pipeline. For instructions, see steps 4 and 5 of Getting started with Continuous Delivery for PE.

1. In the Continuous Delivery for PE web UI, navigate to the control repo or module whose pipelines you want to manage with code.
2. At the top of the **Pipelines** section, click **Manage pipelines**.
3. Select **Manage as code**. Continuous Delivery for PE displays your pipelines in YAML format.
4. Copy the YAML code by scrolling to the bottom of the code block and clicking **Copy to clipboard**.
5. Create a new file named `.cd4pe.yaml` and paste the YAML code into it.

   Important: The file **must** be named `.cd4pe.yaml`.

6. Save the new `.cd4pe.yaml` file to the root directory of your control repo or module, and commit the change in your source control system.

   You can save the `.cd4pe.yaml` file to whichever branch you prefer. In the next step, you'll tell Continuous Delivery for PE where to look for the file in your source control system.
7. In the Continuous Delivery for PE web UI, in the **Select branch** area, select the branch to which you saved the `.cd4pe.yaml` file.
8. Click **Save settings**.

Continuous Delivery for PE now reads the contents of your `.cd4pe.yaml` file before every pipeline run, and uses the YAML code to render the pipelines' definitions in the web UI. Because you're managing your pipelines with code, the pipeline controls in the web UI are disabled.

Update your pipelines using a new `.cd4pe.yaml` file

If you've created pipelines using the Continuous Delivery for PE web UI, but now want to update those pipelines to use new definitions housed in a `.cd4pe.yaml` file, you can do so by saving the file to your control repo or module.

Before you begin

You'll need a properly formatted `.cd4pe.yaml` file containing the definitions of the pipelines you want to create. For more on `.cd4pe.yaml` file syntax, see Structuring a .cd4pe.yaml file.

1. Add a `.cd4pe.yaml` file containing the definitions of the pipelines you want to create to the root directory of your control repo or module. Commit the file to your source control system.

   You can save the `.cd4pe.yaml` file to whichever branch you prefer. Later, you'll tell Continuous Delivery for PE where to look for the file in your source control system.
2. In the Continuous Delivery for PE web UI, navigate to the control repo or module whose pipelines you want to manage with code.
3. At the top of the **Pipelines** section, click **Manage pipelines**.
4. Select **Manage as code**.
5. In the **Select branch** area, select the branch to which you saved the `.cd4pe.yaml` file.
6. Click **Save settings**.

Continuous Delivery for PE now reads to contents of your `.cd4pe.yaml` file before every pipeline run, and uses the YAML code to render the pipelines' definitions in the web UI. Because you're managing your pipelines with code, the pipeline controls in the web UI are disabled.

Create new pipelines using a `.cd4pe.yaml` file

You can create pipelines with code for a brand-new control repo or module in Continuous Delivery for PE. If a `.cd4pe.yaml` file exists in your control repo or module when it is first added to Continuous Delivery for PE, the software will detect this file and ask if you want to use it to build and manage your pipelines.

1. Add a `.cd4pe.yaml` file to the root directory of your control repo or module on the master branch. Commit the file to your source control system.
2. Add the control repo or module to Continuous Delivery for PE.

   For instructions, see steps 4 and 5 of Getting started with Continuous Delivery for PE.
3. When adding the control repo or module, Continuous Delivery for PE will detect the `.cd4pe.yaml` file and ask if you want to use it to build and manage your pipelines. Click **Confirm**.

Continuous Delivery for PE now reads to contents of your `.cd4pe.yaml` file before every pipeline run, and uses the YAML code to render the pipelines' definitions in the web UI. Because you're managing your pipelines with code, the pipeline controls in the web UI are disabled.

Stop managing your pipelines with code

If you want to stop managing your pipelines with a `.cd4pe.yaml` file, and return to using the pipeline controls in the web UI, use the **Manage pipelines** control to make the switch.

1. In the Continuous Delivery for PE web UI, navigate to the control repo or module whose pipelines you are currently managing with a `.cd4pe.yaml` file.
2. At the top of the **Pipelines** section, click **Manage pipelines**.
3. Select **Manage in the web UI** and make any needed adjustments to your pipeline settings.
4. Click **Save settings**.

The pipeline controls in the web UI are enabled. Your `.cd4pe.yaml` file is now ignored by Continuous Delivery for PE, and all pipeline changes must be made in the web UI.

> **Tip:** To avoid confusion, remove the `.cd4pe.yaml` file from your control repo or module.

Structuring a `.cd4pe.yaml` file

When managing your pipelines with code, the pipeline definitions are expressed in a structured format and stored in a file named `.cd4pe.yaml`, which is kept in your control repo or module repo.

Your `.cd4pe.yaml` file must use this structure:

- `spec_version` - The version of the pipelines-as-code file specification you've used to write this YAML file. Currently, only one specification version, `v1`, is available.
- `config` - Pipeline configuration settings for the control repo or module repo as a whole.
- `pipelines` - The names of the pipelines you're creating.
  - `triggers` - What source control activities the pipeline will listen for.
  - `stages` - The parts of the pipeline, and how the pipeline run will advance (or pause and wait for instructions) based on the results of each part.
    - `steps` - All the details about the exact work the pipeline will perform.

Here's an example of a complete `.cd4pe.yaml` file for a control repo. This file describes pipelines for two branches (a master branch and a regex branch).

```
spec_version: v1
config:
  enable_pull_requests_from_forks: false
  deployment_policy_branch: production
```

```
  enable_pe_plans: true
pipelines:
  /feature_.*/:
    triggers:
      - pull_request
      - commit
    stages:
      - name: "Lint/Parser validation"
        auto_promote: all_succeeded
        steps:
          - type: job
            name: control-repo-puppetfile-syntax-validate
          - type: job
            name: control-repo-template-syntax-validate
      - name: "Deploy feature environment"
        steps:
          - type: deployment
            pe_server: cdpe-delivery
            policy: feature_branch
  master:
    triggers:
      - pull_request
      - commit
    stages:
      - name: "Lint/Parser validation"
        auto_promote: all_succeeded
        steps:
          - type: job
            name: control-repo-puppetfile-syntax-validate
          - type: job
            name: control-repo-template-syntax-validate
          - type: job
            name: control-repo-hiera-syntax-validate
          - type: job
            name: control-repo-manifest-validate
      - name: "Impact Analysis"
        auto_promote: any_succeeded
        steps:
          - type: impact_analysis
            concurrent_compilations: 10
            deployments:
              - "Direct merge to Production"
              - "my custom deploy 1"
          - type: pull_request_gate
      - name: "Deploy to lower UAT"
        auto_promote: false
        steps:
          - type: deployment
            policy: direct_merge
            name: "Direct merge to Production"
            target:
              type: node_group
              node_group_id:   fcda068f-e499-44ef-81f2-255fc487b2e2
            pe_server: cdpe-delivery
            parameters:
              stop_if_nodes_fail: 10
              noop: false
              timeout: 60
          - type: deployment
            name: "my custom deploy 1"
            policy:
              source: control-repo
              name: deployments::canary
            target:
```

```
              type: node_group
              node_group_id: fcda068f-e499-44ef-81f2-255fc487b2e2
            pe_server: cdpe-delivery
            parameters:
              noop: true
              max_node_failure: 50
              canary_size: 4
```

Using a `.cd4pe.yaml` file with a module

A `.cd4pe.yaml` file for a module looks almost identical to one for a control repo, with a few important exceptions:

1. If the module pipeline includes an impact analysis step, you must include a pointer to the control repo used in the deployment associated with the impact analysis task.
2. If the `.cd4pe.yaml` file includes a regex branch pipeline using the feature branch deployment policy, your deployment step must include the name of the control repo associated with the module and the control repo branch on which to base the feature branches Continuous Delivery for PE will create.

Here's an example of a complete `.cd4pe.yaml` file for a for a module. The `control_repo` pointer for the impact analysis task is included in the `target:` key section at the bottom of the staging pipeline.

```
spec_version: v1
config:
  enable_pull_requests_from_forks: false
pipelines:
  staging:
    triggers:
      - commit
    stages:
    - name: "Impact analysis"
      steps:
      - type: impact_analysis
        deployments:
        - "Deployment to staging-rustler-1"
        concurrent_compilations: 10
        all_deployments: false
      auto_promote: false
    - name: "Deployment to staging"
      steps:
      - type: deployment
        name: "Deployment to staging-rustler-1"
        policy:
          name: eventual_consistency
        timeout: 3600000
        concurrent_compilations: 0
        all_deployments: false
        pe_server: pe-github
        target:
          type: node_group
          node_group_id: 250fd263-8456-4114-b559-9c6d9fa27748
          control_repo: cdpe-code-rustler-app-2020
```

Here's a module's regex branch pipeline entry using the feature branch deployment policy. The `control_repo` and `base_feature_branch` parameters at the bottom of the `type: deployment` step are required for this special type of pipeline. These parameters specify which control repo is associated with this module, and which branch of that control repo should be used to create feature branches when this pipeline is triggered.

```
pipelines:
 /feature_.*/:
    triggers:
    - commit
    stages:
```

```
    - name: "Validation stage"
      steps:
      - type: job
        name: "module-pdk-validate"
      auto_promote: any_completed
    - name: "Feature branch deployment stage"
      steps:
      - type: deployment
        name: "Deployment to hot-dog-prod-2"
        policy:
          name: "cd4pe_deployments::feature_branch"
        all_deployments: false
        pe_server: "hot-dog-prod-2"
        target:
          type: node_group
        control_repo: "cc-hot-dog-app"
        base_feature_branch: "production"
      auto_promote: false
```

Validating your `.cd4pe.yaml` file

Every time you commit a change to your `.cd4pe.yaml` file in your control repo or module repo, Continuous Delivery for PE uses the YAML code to render the pipelines' definitions in the web UI. To ensure that you commit only well-formed YAML code, use the validation tool in the Continuous Delivery for PE web UI before committing.

1. In the Continuous Delivery for PE web UI, navigate to the control repo or module whose pipelines you're managing with code.
2. At the top of the **Pipelines** section, click **Manage pipelines**.
3. Select **Manage as code**. Continuous Delivery for PE displays your current pipelines in YAML format.
4. Update the code in the window with the changes you wish to make. Alternatively, you can delete the contents of the window and paste in code you've written elsewhere.
5. To check the syntax of your code, click **Validate**.
   - If your syntax is invalid, an error message about the location of the issue appears.
   - If your syntax is valid, **Copy to clipboard** is activated.
6. Once your changes are complete and successfully validated, copy them into the `.cd4pe.yaml` file in your source control or module and commit your change.

`.cd4pe.yaml` file syntax

Your `.cd4pe.yaml` file must be formatted according to the syntax in this section.

Spec version

Every `.cd4pe.yaml` file begins with a `spec_version` declaration expressed as `v<VERSION NUMBER>`. This sets the version of the Continuous Delivery for PE Pipelines as Code specification used when writing the file.

Presently, there is only one version of the Continuous Delivery for PE Pipelines as Code specification, so your file must begin as follows:

```
---
spec_version: v1
```

Config

The `config` section of your `.cd4pe.yaml` file provides global pipeline configuration settings for all the pipelines for the control repo or module where the YAML file is housed.

`config` key	Description	Value	Default
`enable_pull_requests_from_forks`	States whether pull requests from forks of the control repo or module can trigger its pipelines.	Accepts a Boolean.	false
`deployment_policy_branch`	**Optional.** The name of the branch where custom deployment policies are kept. If custom deployment policies are found on the specified branch, they can be used when building pipelines.	Accepts a string.	-
`enable_pe_plans`	**Optional.** States whether to allow the inclusion of Bolt tasks in custom deployment policies used in pipelines for this control repo or module.	Accepts a Boolean.	true

Here's a sample `config` section for a control repo that allows pull requests from forks and serves custom deployment policies from the `production` branch:

```
config:
  enable_pull_requests_from_forks: true
  deployment_policy_branch: production
  enable_pe_plans: true
```

Pipelines

The `pipelines` section names each pipeline you're creating for a control repo or module. Its value is key/value pairs where each key corresponds to a pipeline. The key must have either the same name as a branch or a regular expression (regex).

> **Note:** Only one regular expression entry is allowed per control repo or module.

To create two pipelines (one for a regex branch and one for the master branch), format the `pipelines` section as follows:

```
pipelines:
  /feature_.*/:
 triggers:
   - pull_request
   - commit
 stages:
   - name: "Lint/Parser validation"
       auto_promote: all_succeeded
       steps:
          - type: job
            name: control-repo-puppetfile-syntax-validate

  master:
 triggers:
   - pull_request
   - commit
 stages:
```

```
  - name: "Deploy to production"
      auto_promote: all_succeeded
      steps:
         - type: deployment
        policy: direct_merge
        name: "Direct merge to Production"
        target:
            type: node_group
            node_group_id:   fcda068f-e499-44ef-81f2-255fc487b2e2
        pe_server: cdpe-delivery
        parameters:
            stop_if_nodes_fail: 10
            noop: false
```

Triggers

The `triggers` section must be set for each pipeline you create.

This section specifies the events in your source control system that cause the pipeline to start a run.

The only allowed values are `pull_request` and `commit`. You can set either or both of these values. If no value is set, the pipeline will not run except when triggered manually.

Stages

At least one `stages` section must be set for each pipeline you create.

The `stages` section accepts an array where each item is a stage in the pipeline. Each stage has a name, instructions about when and whether to auto-promote to the next stage in the pipeline, and a list of steps to be executed in parallel.

`stages` key	**Description**	**Value**	**Default**
`name`	The name you're giving the pipeline stage.	Accepts a string in quotes.	Stage <STAGE NUMBER>

`stages` key	Description	Value	Default
`auto_promote`	Instructions on whether or not to automatically go on to the next pipeline stage based on the conditions specified.	Accepts one of the following: • `false` - Only manual promotions allowed. • `all_succeeded` - Auto-promote only if all steps succeed. • `any_succeeded` - Auto-promote if any step succeeds. • `all_completed` - Auto-promote only if all steps complete (succeed or fail) and no steps are canceled. • `any_completed` - Auto-promote if any step completes (succeeds or fails) and not all steps are canceled.	`all_succeeded`

Steps

At least one `step` section must be set for each `stages` section you create.

A step defines a particular job to be performed in a pipeline stage. All steps are run concurrently, so the order in which you list steps in a stage doesn't matter.

Every step is defined using a hash of key/value pairs. The `type` key is always required, and the other keys are dependent on the type of step selected. The `type` key accepts the following values:

`type` values	Description
`job`	A named test for Puppet code. The jobs available to your workspace are found by clicking **Jobs** in the navigation bar in the web UI.
`impact_analysis`	A report assessing the code change in the pipeline for potential impact to nodes and configurations in deployments defined later in the pipeline.

`type` values	Description
`pull_request_gate`	A conditional checkpoint that ends the pipeline execution at the pull request gate if the pipeline was triggered by a pull request. If the pipeline was triggered by a commit, the pipeline execution will continue beyond the pull request gate.
`deployment`	A Puppet code deployment.

Keys for `type: job`

`job` key	Description	Required?	Default
`name`	The name of the job as listed in the **Jobs** page for the workspace.	yes	-

To add a job called `control-repo-puppetfile-syntax-validate` to a pipeline, format the `type: job` entry as follows:

```
steps:
  - type: job
    name: control-repo-puppetfile-syntax-validate
```

Keys for `type: impact_analysis`

> **Note:** If you include an impact analysis step in a `.cd4pe.yaml` file for a module, see the instructions in the **Keys for `type: deployment`** section about setting `control_repo` on the deployment associated with the imapct analysis task.

`impact_analysis` key	Description	Required?	Default
`concurrent_compilations`	How many catalog compilations to perform at the same time. **Tip:** If your compilers are hitting capacity when performing an impact analysis, lower this number. Lowering this number increases the impact analysis run time.	no	10
`deployments`	A list of the deployments to assess the potential impact. Accepts an array of strings where each item is the name of a deployment in your pipeline definition.	You must set either `deployments` or `all_deployments`.	-

`impact_analysis` key	Description	Required?	Default
`all_deployments`	Whether to assess the impact to all deployments in the pipeline. Accepts a Boolean value.		false
`puppetdb_connection_timeout_sec`	The timeout period (in seconds) for requests to PuppetDB during an impact analysis task.	no	120

To add an impact analysis task running 10 compilations at a time on two specific deployments, and a second impact analysis task running on all deployments in your pipeline that times out PuppetDB requests after three minutes, form the `type: impact_analysis` entries as follows:

```
steps:
  - type: impact_analysis
    concurrent_compilations: 10
    deployments:
      - "Direct merge to Production"
      - "My PCI deployment"
  - type: impact_analysis
    all_deployments: true
    puppetdb_connection_timeout_sec: 180
```

Keys for `type: pull_request_gate`

The `pull_request_gate` step type does not take any additional keys.

To add a pull request gate to your pipeline, format the `type: pull_request_gate` entry as follows:

```
steps:
  - type: pull_request_gate
```

Keys for `type: deployment`

`deployment` key	Description	Value	Required?
`policy`	The deployment policy to be used.	For built-in deployment policies, accepts the policy name as a string. For custom deployment policies, accepts a key-value pair defining the policy to be used.	yes
`name`	The name you're giving the deployment.	Accepts a string in quotes.	yes

`deployment` key	Description	Value	Required?
`target`	The infrastructure (node group) that the deployment will target. The `target` key is not used for deployments from regex branches.	Accepts key/value pairs and requires a `type: node_group` key. If used for a module pipeline that contains an impact analysis task, also requires a `control_repo:` key. **Note:** See **Formatting the `target` key** below.	yes, except for regex branches
`pe_server`	The name in Continuous Delivery for PE of the Puppet Enterprise instance the deployment will target.	Accepts a string. **Tip:** Find the name of your PE instance by going to **Settings** > **Puppet Enterprise** in the web UI.	yes
`parameters`	Any relevant parameters defined by the deployment policy. To find out which parameters the policy takes, see Built-in deployment policies or refer to the custom deployment policy's documentation.	Accepts key/value pairs.	no
`control_repo`	The name of the control repo a module is associated with.	Accepts a string.	only for module regex branch pipelines using the feature branch deployment policy
`base_feature_branch`	The branch in the `control_repo` that Continuous Delivery for PE will use to create feature branches.	Accepts a string.	only for module regex branch pipelines using the feature branch deployment policy

To add a deployment using a built-in deployment policy to your pipeline, format the `type: deployment` entry as follows:

```
steps:
  - type: deployment
    policy: cd4pe_deployments::direct_merge
    name: "Direct merge to Production"
    target:
      type: node_group
      node_group_id:  fcda068f-e499-44ef-81f2-255fc487b2e2
    pe_server: cdpe-delivery
    parameters:
      stop_if_nodes_fail: 10
      noop: false
```

```
      timeout: 60
```

To add a deployment using a custom deployment policy to your pipeline, format the `type: deployment` entry as follows:

```
steps:
  - type: deployment
    policy:
      name: deployments::custom_policy1
      source: name-of-control-repo
    name: "Direct merge to Production"
    target:
      type: node_group
      node_group_id:  fcda068f-e499-44ef-81f2-255fc487b2e2
    pe_server: cdpe-delivery
    parameters:
      policy_parameter1: 10
      policy_parameeter2: false
```

CAUTION: Do not include sensitive parameters in your custom deployment policy if you use the custom deployment policy in a pipeline managed with code.

To add a deployment to a module for a regex branch pipeline using the feature branch deployment policy, format the type: deployment entry as follows:

```
steps:
  - type: deployment
    policy:
      name: "cd4pe_deployments::feature_branch"
    name: "Deployment to hot-dog-prod-2"
    target:
          type: node_group
    pe_server: "hot-dog-prod-2"
    control_repo: "cc-hot-dog-app"
    base_feature_branch: "production"
```

Formatting the `target` key

Note: The `target` key is not used for deployments from regex branches.

When used for a **control repo** pipeline or a module pipeline that does not include impact analysis, the `target` key must be formatted as follows:

```
    target:
      type: node_group
      node_group_id:  fcda068f-e499-44ef-81f2-255fc487b2e2
```

Presently, `node_group` is the only available `type`.

To locate your target node group's ID:

1. In the PE console, click **Classification**.
2. Locate your target node group and click its name. A page with information about the node group opens.
3. In the URL for the page, locate and copy the alphanumeric string that follows `/groups/`. This is your node group ID.

When used for a **module** pipeline that includes an impact analysis step, the `target` key must be formatted as follows:

```
    target:
```

```
      type: node_group
      node_group_id:  fcda068f-e499-44ef-81f2-255fc487b2e2
      control_repo:  cdpe-2018-1-pe-master-1-control
```

The `control_repo` key's value is the name of the control repo used in the deployment associated with the impact analysis task.

Analyzing the impact of code changes

Continuous Delivery for Puppet Enterprise (PE) lets you see the potential impact that new Puppet code will have on your PE-managed infrastructure even before the new code is merged. You can generate an impact analysis report on demand for any commit to a control repo or module, or add impact analysis to a pipeline to automatically generate a report on all proposed code changes.

- Generating impact analysis reports on page 109

Impact analysis reports show you the effect and risk of proposed code changes, allowing for quick review of lower-risk changes while enabling additional scrutiny for higher-risk changes. Add an impact analysis task to each control repo and module repo pipeline to generate an impact analysis report for every change submitted to your repos.

- Limitations of impact analysis on page 111

Some code changes may impact your infrastructure beyond what an impact analysis report displays. The technical limitations of impact analysis reports are outlined here.

Generating impact analysis reports

Impact analysis reports show you the effect and risk of proposed code changes, allowing for quick review of lower-risk changes while enabling additional scrutiny for higher-risk changes. Add an impact analysis task to each control repo and module repo pipeline to generate an impact analysis report for every change submitted to your repos.

> **Note:** When impact analysis is enabled, catalog compiles are generated every time an impact analysis task detects that a node may be impacted by a change to your Puppet code. These additional compiles increase the performance load placed on your PE master.

Add impact analysis to a module pipeline

Once you add an impact analysis task to your module pipeline, the impact analysis report is automatically generated each time the pipeline is triggered and the conditions you've set are met.

Before you begin

1. Configure impact analysis.
2. Set up a pipeline for your module that includes at least one deployment.

> **CAUTION:** Impact analysis fails if you include the `$environment` variable in your Puppet manifest. Instead, use Hiera and class parameters.

Impact analysis reports are generated by diffing a newly generated catalog for the deployment conditions you've set against the current catalog for the same deployment. The results of this process are shown in the impact analysis report.

You can add as many impact analysis tasks to your pipeline as you wish, but each stage in the pipeline can have only one impact analysis task. An impact analysis task cannot be in the same stage as a deployment.

1. In the Continuous Delivery for PE web UI, click **Modules**. Click the name of the module you wish to add impact analysis to.

2. Select the pipeline you wish to add impact analysis to. Make sure there is at least one deployment present in the pipeline.

   Impact analysis is calculated for your pipeline using the deployment conditions you've chosen.

3. Add impact analysis to a stage that does not contain a deployment. Click **More actions** . Select **Add item to stage**, then select **Impact analysis**.

   > **Tip:** For best results, add impact analysis to a stage in your pipeline that comes before a PR gate and before any deployment stages.

4. Optional: Set the catalog compilation batch size. By default, Continuous Delivery for PE compiles 10 catalogs at a time when performing an impact analysis task.
5. Select the environments you want to generate an impact analysis report for.
6. For each selected environment, choose the control repo where the code associated with that environment is stored.
7. Click **Add impact analysis**.

Impact analysis is now enabled for your module pipeline. An impact analysis report is generated each time the pipeline runs.

Add impact analysis to a control repo pipeline

Once you add an impact analysis task to your control repo pipeline, the impact analysis report is automatically generated each time the pipeline is triggered and the conditions you've set are met.

Before you begin

1. Configure impact analysis.
2. Set up a pipeline for your control repo that includes at least one deployment.

> **CAUTION:** Impact analysis fails if you include the `$environment` variable in your Puppet manifest. Instead, use Hiera and class parameters.

Impact analysis reports are generated by diffing a newly generated catalog for the deployment conditions you've set against the current catalog for the same deployment. The results of this process are shown in the impact analysis report.

You can add as many impact analysis tasks to your pipeline as you wish, but each stage in the pipeline can have only one impact analysis task. An impact analysis task cannot be in the same stage as a deployment.

1. In the Continuous Delivery for PE web UI, click **Control repos**. Click the name of the control repo you wish to add impact analysis to.
2. Select the pipeline you wish to add impact analysis to. Make sure there is at least one deployment present in the pipeline.

   Impact analysis is calculated for your pipeline using the deployment conditions you've chosen.

3. Add impact analysis to a stage that does not contain a deployment. Click **More actions** . Select **Add item to stage**, then select **Impact analysis**.

   > **Tip:** For best results, add impact analysis to a stage in your pipeline that comes before a PR gate and before any deployment stages.

4. Optional: Set the catalog compilation batch size. By default, Continuous Delivery for PE compiles 10 catalogs at a time when performing an impact analysis task.

5. Determine which environments you want to generate an impact analysis report for.
   - Select **Run for all environments in the pipeline** to generate impact analysis for all environments used by all deployments in the pipeline.
   - Select **Run for selected environments** to choose which environments to run impact analysis on from among those used by the pipeline's deployments.
6. Click **Add impact analysis**.

Impact analysis is now enabled for your control repo pipeline. An impact analysis report is generated each time the pipeline runs.

Generate an impact analysis report on demand

You can create a new impact analysis report for any code change committed to a control repo or module in Continuous Delivery for PE without triggering a pipeline.

Before you begin

1. Configure impact analysis.

Impact analysis reports are generated by diffing a newly generated catalog for the deployment conditions you've set against the current catalog for the same deployment. The results of this process are shown in the impact analysis report.

1. In the Continuous Delivery for PE web UI, navigate to the control repo or module you wish to generate an impact analysis report for.
2. Click **Manual actions** and select **New impact analysis**.
3. Select the branch where the code you want to analyze is located, then select the specific commit that contains the new code.
4. Select the PE instance that manages the nodes you want to analyze, then select the specific node group.
   - For impact analysis on a **control repo**: If relevant, select an environment prefix.
   - For impact analysis on a **module**: Select the control repo where the module's environment code is deployed.
5. Finally, set the number of node catalogs that should be compiled concurrently.
6. Click **Analyze**. A new impact analysis report based on your selections is generated for you. Click **View impact analysis** to be redirected to the report.

> **Note:** Depending on the size of your node group, it may take several minutes for the impact analysis report to be created.

Limitations of impact analysis

Some code changes may impact your infrastructure beyond what an impact analysis report displays. The technical limitations of impact analysis reports are outlined here.

We've designed impact analysis to give you helpful information about the potential results of code changes to your infrastructure. However, impact analysis is not designed to be an exhaustive report — do not use it as a substitute for more exhaustive testing. For instance, there are certain circumstances in which we can't reliably calculate the full impact of a code change. As an impact analysis user, it's critical that you understand that these limitations exist, and that you remain alert to the possibility that code changes might have consequences for your infrastructure beyond what impact analysis shows.

The following is a non-exhaustive list of some of the limitations of impact analysis. This list might be updated from time to time, but is not complete.

- **Changes to an environment's `environment.conf` file.** The downstream impact of changes to an `environment.conf` file cannot be analyzed.

- **Brand-new code.** New Puppet classes, facts, or functions that have never before been applied to any of your nodes cannot be analyzed.
- **Changes to functions.** The impact of changes to a function cannot be analyzed.
- **Changes to facts.** The impact of changes to a fact cannot be analyzed.
- **Changes to class names used in classification.** Changes to the names of classes that have been previously classified in the classifier cannot be analyzed. A classification update in the classifier is required before Puppet can locate the newly renamed class.
- **Imported resources.** While impact analysis can provide information about nodes that export resources, the impact to nodes that receive those exported resources cannot be analyzed.
- **Sensitive data types.** Any changes to data types marked as sensitive will not be analyzed.
- **Alias metaparameter.** Any resources using the alias metaparameter cannot be analyzed.
- **Changes to Embedded Puppet templates.** The impact of changes made to Embedded Puppet (`.epp`) or Embedded Ruby (`.erb`) template files alone cannot be analyzed.

> **Remember:** Reports generated by impact analysis are provided "AS-IS."

Deploying Puppet code

Built-in deployment policies

Deployment policies are prescriptive workflows for Puppet code deployment that are built into Continuous Delivery for Puppet Enterprise (PE). You select the best deployment policy for your situation, and Continuous Delivery for PE does all the Git heavy lifting for you, deploying your code to the right nodes.

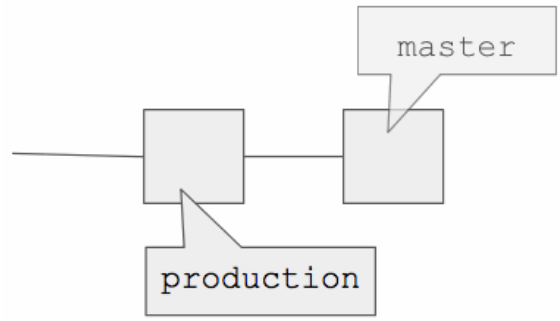
Which deployment policy should I use?

Continuous Delivery for PE includes built-in deployment policies, each with a different branching workflow. Use the table below to select the best deployment policy for your circumstances.

Deployment policy	Best for...
Direct deployment policy	• Small or trivial changes • Changes that you have a high level of confidence will not cause issues when deployed
Temporary branch policy	• Fully tested changes • Changes you'd like to have validated through deployment • Workflows that don't require extensive logging of historic data
Eventual consistency policy	• Fully tested changes • Large PE installations with regularly scheduled Puppet runs • Situations in which running additional orchestrator jobs is undesirable
Feature branch policy	• Changes made on a feature branch that haven't been merged

Direct deployment policy

The direct deployment policy is the most basic of the four deployment policies offered in Continuous Delivery for PE. This policy is best to use if you wish to deploy a certain commit to a specified environment, then run Puppet on that environment (and only that environment) to deploy your changes.



When you kick off a deployment using the direct deployment policy, Continuous Delivery for PE performs the following steps.

Step 1: `puppet-code deploy`

Using Code Manager, Continuous Delivery for PE synchronizes the selected code change with your PE instance by running `puppet-code deploy` on Puppet Server.

Step 2: Temporary environment group

Continuous Delivery for PE creates a new environment group that's a child of the node group you selected when setting up your deployment. The child node group inherits all the rules, configuration settings, and variables of its parent node group. A Puppet Query Language query pins all the nodes associated with the parent node group to the child node group.

This environment node group is temporary; when the deployment is complete, it is automatically deleted. It's likely that you won't ever need to interact directly with this environment group.

Step 3: Running Puppet

Continuous Delivery for PE uses the orchestrator to kick off a Puppet run on the nodes in the child node group. The orchestrated Puppet run updates nodes as quickly as the concurrency limits of Puppet Server allow. You can monitor the progress of this process on the deployment details page in the Continuous Delivery for PE web UI.

Step 4: Clean-up

When the Puppet run is complete, Continuous Delivery for PE deletes the child environment group and moves the HEAD of your target branch in your source control repository to the commit you chose to deploy.

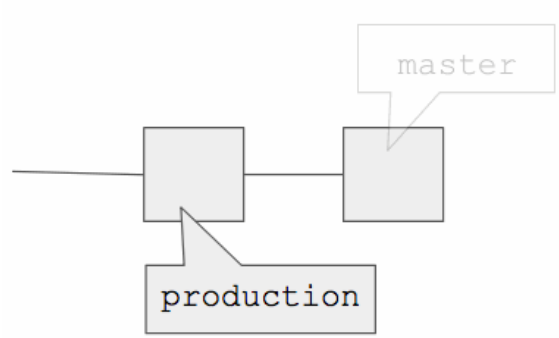
Direct deployment policy parameters

The following optional parameters are available to customize the behavior of your direct deployment. Set these parameters in the web UI when creating a new deployment using the direct deployment policy, or add them to the your `.cd4pe.yaml` file as part of a `deployment` step type declaration.

Parameter	Description	Required?	Default value	Value type
`max_node_failure`	The maximum number of nodes that can fail before the deployment is stopped and reported as a failure.	no	-	integer
`noop`	Whether to perform the Puppet runs in no-op mode.	no	false	Boolean
`fail_if_no_nodes`	Whether to stop the deployment and report a failure if the selected node group does not contain any nodes.	no	false	Boolean

Temporary branch policy

During a deployment using the temporary branch policy, Continuous Delivery for PE creates a temporary Git branch containing the code you wish to deploy, and a temporary environment group containing the nodes you're deploying code to. Your new code is then mapped to the temporary environment group in batches, allowing you to be certain that your new Puppet code works with each node before the changes are deployed to the full environment.



When you kick off a deployment using the temporary branch policy, Continuous Delivery for PE performs the following steps.

Step 1: New branch

Continuous Delivery for PE creates a new branch in your source control repository with the commit you've selected at its tip. This branch is temporary; when the deployment is complete, it will be automatically deleted. It's likely that you won't ever need to interact directly with this temporary branch.

Step 2: `puppet-code deploy`

Using Code Manager, Continuous Delivery for PE synchronizes the code changes with your PE instance by running `puppet-code deploy` on Puppet Server.

Step 3: Temporary environment group

Continuous Delivery for PE creates a new environment group that's a child of the node group you selected when setting up your deployment. The child node group inherits all the rules, configuration settings, and variables of its parent node group. A Puppet Query Language query pins all the nodes associated with the parent node group to the child node group.

This environment node group is temporary; when the deployment is complete, it is automatically deleted. It's likely that you won't ever need to interact directly with this environment group.

Step 4: Running Puppet

Continuous Delivery for PE uses the orchestrator to kick off a Puppet run on the nodes in the child node group. Puppet runs on a few nodes at a time, based on the stagger settings you specified when setting up the deployment. You can monitor the progress of this process on the deployment details page in the Continuous Delivery for PE web UI.

Step 5: Clean-up

When all the Puppet runs have completed successfully (or in the event that the termination conditions you specified are met), Continuous Delivery for PE deletes the child environment group, moves the HEAD of the target branch in your source control repository to point to your chosen commit, and deletes the temporary Git branch created at the beginning of this process.

Temporary branch deployment policy parameters

The following optional parameters are available to customize the behavior of your temporary branch deployment. Set these parameters in the web UI when creating a new deployment using the temporary branch deployment policy, or add them to the your `.cd4pe.yaml` file as part of a `deployment` step type declaration.

Parameter	Description	Required?	Default value	Value type
`batch_delay`	How many seconds to wait between each deployment batch.	no	60	integer
`batch_size`	How many nodes to include in a deployment batch.	no	10	integer
`max_node_failure`	The maximum number of nodes that can fail before the deployment is stopped and reported as a failure.	no	-	integer
`noop`	Whether to perform the Puppet runs in no-op mode.	no	false	Boolean
`fail_if_no_nodes`	Whether the deployment should fail if there are no nodes in the target node group or environment.	no	true	Boolean

Eventual consistency policy

The eventual consistency policy is designed for situations in which you wish to deploy new Puppet code, but prefer to wait for nodes to check in on their regular schedule to pick up the changes, rather than kicking off a dedicated Puppet run with the orchestrator. By using this policy, impacted nodes all eventually reach consistency with the new code.

When you kick off a deployment using the eventual consistency policy, Continuous Delivery for PE performs the following steps:

Step 1: `puppet-code deploy`

Using Code Manager, Continuous Delivery for PE synchronizes the selected code change with your PE instance by running `puppet-code deploy` on Puppet Server.

Step 2: Code changes are implemented during the next round of scheduled Puppet runs

The new code is delivered to the impacted nodes the next time they check in according to their established schedule.

Eventual consistency deployment policy parameters

The eventual consistency deployment policy does not have any optional or required policy parameters.

Feature branch deployment policy

The feature branch deployment policy is used only in regex branch pipelines. Code changes you've made in a feature branch are deployed to a Puppet environment with the same name as the feature branch.

When you kick off a deployment to a regex branch pipeline using the feature branch deployment policy, Continuous Delivery for PE performs the following steps:

Step 1: `puppet-code deploy`

Using Code Manager, Continuous Delivery for PE deploys the commit in the pipeline to a Puppet environment with the same name as the branch that triggered the pipeline. If an environment with this name doesn't already exist, Continuous Delivery for PE creates it during the deployment.

Feature branch deployment policy parameters

The feature branch deployment policy does not have any optional or required policy parameters.

Creating custom deployment policies

If the built-in deployment policies included in Continuous Delivery for PE don't align with the way your organization works with Puppet Enterprise, you can write a custom deployment policy tailored to your needs.

CAUTION: Custom deployment policies are a beta feature. As such, they may not be fully documented or work as expected; please explore them at your own risk.

In a custom deployment policy, you declare the steps that need to happen in order to deploy your Puppet code changes. Because custom deployment policies leverage Bolt plans, you can configure your policies to automate the processes that make up your current deployment workflow. This means your custom policies can include:

- Git processes in your source control provider of choice
- Changes to your node groups with Puppet orchestrator
- Bolt tasks and plans
- Automatic notifications via the third-party services you use daily, such as Slack and ServiceNow

The `puppetlabs-cd4pe_deployments` module is built into Continuous Delivery for PE, so you don't need to download or install anything in order to use custom deployment policies. To see the full list of what's available for inclusion in your custom deployment policy, see the documentation for the `puppetlabs-cd4pe_deployments` module.

If you need some inspiration, see how the deployment policies built into Continuous Delivery for PE look when written in custom deployment policy format.

Custom deployment policy prerequisites and cautions

- In order for your custom deployment policies to be available for use in the web UI, you must have webhooks enabled between Continuous Delivery for PE and your source control system.
- Do not include sensitive parameters in your custom deployment policy if you use the custom deployment policy in a pipeline managed with code.

Add a custom deployment policy to your control repo

Each custom deployment policy file you create must be stored in a module named `deployments` in the control repo that will utilize the policy.

Before you begin

Decide which branch in your control repo will house your custom deployment policies. By default, Continuous Delivery for PE looks for custom deployment policies on the `production` branch.

If your control repo doesn't have a `production` branch, or if you prefer to store custom deployment policies on a different branch, tell Continuous Delivery for PE where to look for your custom deployment policies by using the `deployment_policy_branch` setting of the `config` section of `.cd4pe.yaml` file. For details, see Structuring a .cd4pe.yaml file.

Important: If you don't store your custom deployment policies on the `production` branch, you must manage your control repo's pipelines with a `.cd4pe.yaml` file in order to use them.

1. In your control repo, on the `production` branch (or the branch you've set as the `deployment_policy_branch` in your `.cd4pe.yaml` file), create a new Puppet module named `deployments`.

   In order for Continuous Delivery for PE to find and run your custom deployment policies correctly, this module must live in the `/modules`, `/site`, or `/site-modules` directory of your control repo.
2. Add your custom deployment policy file to the `/plans` directory in the `deployments` module.

Deploy code manually

Use the manual deployment workflow to push a code change to a specified group of nodes on demand.

1. In the Continuous Delivery for PE web UI, click **Control repos**, then select the control repo you wish to deploy from.
2. Click **Manual actions** and select **New deployment**.
3. Select the branch on which you made the change you're going to deploy, the commit you want to deploy, and your Puppet Enterprise instance.

   > **Note:** If you're creating an on-demand deployment for a regex branch pipeline, the commit at the HEAD of your feature branch is selected for you.
4. Select the Puppet environment you wish to deploy the change to.
5. Select a deployment policy.

   See Deployment policies to learn more about the deployment policies and how they work.
6. Optional: Set termination conditions for this deployment, and choose the number of nodes that can fail before the deployment is stopped.
7. Give the deployment a name, then click **Deploy**.

   Monitor the progress of your deployment on the deployment details page that opens when you launch the deployment.

Deploy module code

You can deploy new module code to your Puppet environments via a Continuous Delivery for PE module pipeline. To do so, you must first add a `:branch => :control_branch` declaration to the module's entry in your control repo's Puppetfile.

Module code is deployed to your Puppet environments using the eventual consistency deployment policy. When you trigger a module deployment, Continuous Delivery for PE creates a new branch in your module repository with the same name as your target Puppet environment. This new branch contains the code to be deployed.

Continuous Delivery for PE then triggers Code Manager, which reads the `:branch => :control_branch` declaration referring to the module in the control repo's Puppetfile and adds the new module code to the control repo. The new module code is now ready to be deployed to your chosen Puppet environments on the next scheduled Puppet run.

1. Open the Puppetfile that includes the module you wish to deploy. Add a `:branch => :control_branch` declaration to the module's section of the Puppetfile, as in the following example.

   ```
   mod 'apache',
     :git    => 'https://github.com/puppetlabs/puppetlabs-apache',
     :branch => :control_branch,
     :default_branch => 'master'
   ```

   To learn more about the `:branch => :control_branch` option, see Declare content from a relative control repo branch in the Code Manager documentation.

2. In the Continuous Delivery for PE web UI, click **Modules** and select the module you wish to deploy.
3. If you haven't already done so, create a pipeline for your module.
4. Click **+ Add stage**. Select **Module deployment**.
5. Select your PE instance and choose the Puppet environment the module code will be deployed to.
6. Click **Add deployment**.

Your module pipeline is now set up to deploy new module code to your chosen Puppet environments any time the pipeline is triggered.

Require approval for deployments to protected Puppet environments

If your organization's business processes require manual review and approval before Puppet code is deployed to certain environments, set up an approval group of individuals with the authority to provide the needed review and sign-off. These approvers are contacted each time a deployment to a protected environment is proposed.

Before you begin
Make sure SMTP has been configured for your Continuous Delivery for PE installation. For instructions, see Configure SMTP.

Enabling a manual approval checkpoint on deployments to protected Puppet environments is a two-step process. First, designate the Continuous Delivery for PE users with the authority to approve or reject deployment requests. Next, designate the Puppet environments that require manual deployment approval.

Important: Designating approvers requires super user permissions.

1. Create an approval group. The members of this group review all proposed deployments to the environments you designate as protected and manually approve or decline each deployment.
   a) In the Continuous Delivery for PE web UI, click **Settings**.
   b) In the **Groups** tab, click **+ Create new group**.
   c) Enter a name (such as `Approval`) and description for your new group, then click **Save**.
   d) In each permissions category, select the permissions you wish to assign to the approval group and click **Save and add users**.

      Important: At a minimum, the approval group must have the **List** permission for **Control repos** in order to view and approve or deny deployments.

   e) On the **Add users** page, add the individuals with the authority to approve or deny deployments to protected environments.

      - Search for users by username or email address.
2. Designate which Puppet environments require deployment approval.
   a) Click the **Puppet Enterprise** tab.
   b) Click the number (likely "0") in the **Protected environments** column for your PE instance.
   c) Select the Puppet environment that requires deployment approval.
   d) Select the approval group you created in step 1.
   e) Click **Add**.
   f) If necessary, repeat these steps to designate additional environments as protected, then click **Done**.

Now that this set-up process is complete, each time a deployment to the protected environment is triggered, either manually or through a pipeline run, the members of the approval group receive an email and a message in the message center alerting them that approval of the deployment is required.

A member of the approval group must review the deployment's details page and click **Provide approval decision**. After they approve or decline the deployment, a record of their decision is added to the deployment's details page.

Troubleshooting

Use this guide to troubleshoot issues with your Continuous Delivery for Puppet Enterprise (PE) installation.

> **Note: For PE instances with a replica configured for disaster recovery.** In the event of a partial failover, Continuous Delivery for PE is not available. Learn more at What happens during failovers in the PE documentation. To restore Continuous Delivery for PE functionality, you must promote the replica to primary server.

Stopping and restarting the Continuous Delivery for PE container

Continuous Delivery for PE is run as a Docker container, so stopping and restarting the container is an appropriate first step when troubleshooting.

If you installed Continuous Delivery for PE with the `cd4pe` module:

- Stop the container: `service docker-cd4pe stop`
- Restart the container: `service docker-cd4pe start`

If you installed Continuous Delivery for PE from the PE console **and** installed the `cd4pe` module to automate upgrades:

- Stop the container: `service docker-cd4pe stop`
- Restart the container: `service docker-cd4pe start`

If you installed Continuous Delivery for PE using Docker, or if you installed from the PE console but have not installed the `cd4pe` module:

- Stop the container: `docker stop <CONTAINER_NAME>`
- Restart the container: `docker run <CONTAINER_NAME>`, passing in any applicable environment variables

> **Note:** To print the name of the container, run `docker ps`.

Accessing the Continuous Delivery for PE logs

Because Continuous Delivery for PE is run as a container in Docker, the software's logs are housed in Docker.

To access the Continuous Delivery for PE logs, run `docker logs <CONTAINER_NAME>`.

> **Note:** To print the name of the container, run `docker ps`.

If you installed the software using the `cd4pe` module, run `docker logs cd4pe`.

PE component errors in Docker logs

The Docker logs include errors for both Continuous Delivery for PE and for the numerous PE components used by the software. It’s important to realize that an error in the Continuous Delivery for PE Docker log may actually indicate an issue with Code Manager, r10k, or another component running outside the Docker container.

For example, this Docker log output indicates a Code Manager issue:

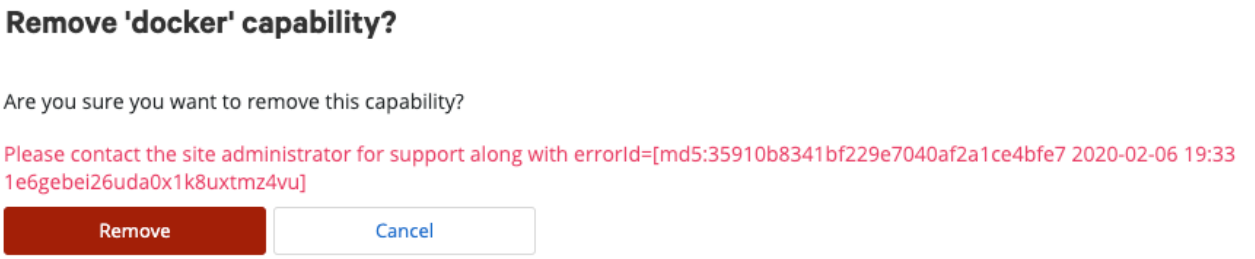
```
Module Deployment failed for
 PEModuleDeploymentEnvironment[nodeGroupBranch=cd4pe_lab,
nodeGroupId=a923c759-3aa3-43ce-968a-f1352691ca02, nodeGroupName=Lab
 environment,
peCredentialsId=PuppetEnterpriseCredentialsId[domain=d3, name=lab-MoM],
```

```
pipelineDestinationId=null, targetControlRepo=null,
 targetControlRepoBranchName=null,
targetControlRepoHost=null, targetControlRepoId=null].
Puppet Code Deploy failure: Errors while deploying environment
 'cd4pe_lab' (exit code: 1):
ERROR -> Unable to determine current branches for Git source 'puppet' (/etc/
puppetlabs/code-staging/environments)
Original exception: malformed URL 'ssh://git@bitbucket.org:mycompany/
control_lab.git'
at /opt/puppetlabs/server/data/code-manager/worker-caches/deploy-pool-3/
ssh---git@bitbucket.org-mycompany-control_lab.git
```

For help resolving issues with PE components, see the Puppet Enterprise troubleshooting documentation.

Error IDs in web UI error messages

Occasionally, error messages shown in the Continuous Delivery for PE web UI include an error ID and instructions to contact the site administrator. For example:



Errors of this kind are shown without additional detail for security purposes. Users with root access to the Continuous Delivery for PE host system can search the Docker log for the error ID to learn more.

Duplicate job logs after reinstall

A job's log is housed in object storage after the job is complete. If you reinstall Continuous Delivery for PE but reuse the same object storage without clearing it, you might notice logs for multiple jobs with the same job number, or job logs already present when a new job has just started.

To remove duplicate job logs and prevent their creation, make sure you clear both the object storage and the database when reinstalling Continuous Delivery for PE.

Name resolution

Inside its Docker container, Continuous Delivery for PE relies on its host's DNS system for name resolution. Many `can't reach` and `timeout connecting` errors reported in the Docker logs are actually DNS lookup failures.

To address name resolution in Continuous Delivery for PE, you have two options:

1. Set up a DNS server for Continuous Delivery for PE that includes all the required hosts.
2. Use the `cd4pe_docker_extra_params` flag to add hostnames to the container's `/etc/hosts` file. The value will be an array of strings in the format `--add-host <HOSTNAME>:<IP_ADDRESS>`. For example:

   ```
   ["--add-host pe-201920-master.mycompany.vlan:10.234.3.142", "--
   add-host pe-201920-agent.mycompany.vlan:10.234.3.110", "--add-host
    gitlab.mycompany.vlan:10.32.3.110"]
   ```

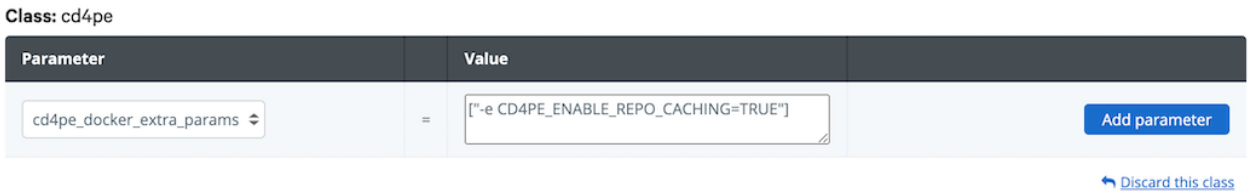
   See Advanced configuration options for more information.

If you need a temporary workaround to solve a name resolution issue, you can edit the Docker container's `/etc/hosts` file directly. However, these changes will be erased whenever the Docker container restarts, so this method should not be used as a permanent solution.

Improving job performance by caching Git repositories

Users with large Git repositories can enable Git repository caching in order to improve job performance. By default, repository caching is disabled.

To enable Git repository caching, set `CD4PE_ENABLE_REPO_CACHING=TRUE` as a value for the `cd4pe_docker_extra_params` parameter on the `cd4pe` class.



Note: To pass additional arguments to the Docker process running the Continuous Delivery for PE container, you must specify them as an array. For example: `["--add-host gitlab.puppetdebug.vlan:10.32.47.33","-v /etc/puppetlabs/cd4pe/config:/config","--env-file /etc/puppetlabs/cd4pe/env-extra","-e CD4PE_LDAP_GROUP_SEARCH_SIZE_LIMIT=250"]`

Including the `.git` directory in cached repositories

Beginning with version 3.12.0, the `.git` directory is automatically omitted when copying cached Git repositories to job hardware. This means that the job cannot perform Git actions on the code. If needed, you can adjust this setting so that the `.git` directory is included in the cached repository.

To include the `.git` directory in copies of cached Git repositories sent to job hardware, set `CD4PE_INCLUDE_GIT_HISTORY_FOR_CD4PE_JOBS=TRUE` as a value for the `cd4pe_docker_extra_params` parameter on the `cd4pe` class.

Adjusting the timeout period for jobs

In some circumstances, such as when working with large Git repositories, you may need to adjust the length of the job timeout period. Use the environment variables in this section to customize your job timeout period.

To use any of the environment variables in this section, set them as a value for the `cd4pe_docker_extra_params` parameter on the `cd4pe` class.

Note: To pass additional arguments to the Docker process running the Continuous Delivery for PE container, you must specify them as an array. For example: `["--add-host gitlab.puppetdebug.vlan:10.32.47.33","-v /etc/puppetlabs/cd4pe/config:/config","--env-file /etc/puppetlabs/cd4pe/env-extra","-e CD4PE_LDAP_GROUP_SEARCH_SIZE_LIMIT=250"]`

Variable	Description	Default value
`CD4PE_JOB_GLOBAL_TIMEOUT_MINUTES`	Sets the default timeout period for jobs. Once this timeout period elapses, the job fails and a timeout message is printed to the log.	12 (minutes)

Variable	Description	Default value
`CD4PE_JOB_HTTP_READ_TIMEOUT_MINUTES`	Sets the timeout period for a job connecting to an endpoint.	11 (minutes)
`CD4PE_REPO_CACHE_RETRIEVAL_TIMEOUT_MINUTES`	**Only used if Git repository caching is enabled.** Sets the timeout period for a thread attempting to access a cached Git repository.	10 (minutes)
`CD4PE_BOLT_PCP_READ_TIMEOUT_SEC`	Sets the Bolt PCP read timeout period. **Note:** Jobs cannot proceed while file sync is running. If a file sync operation is not completed before the Bolt PCP read timeout period elapses, the job fails. Increase the Bolt PCP read timeout period to prevent these job failures.	60 (seconds)

In order to ensure that all useful error messages are printed to the logs, make sure that the value of `CD4PE_JOB_GLOBAL_TIMEOUT_MINUTES` is larger than the value of `CD4PE_JOB_HTTP_READ_TIMEOUT_MINUTES`, and that both are larger than the value of `CD4PE_REPO_CACHE_RETRIEVAL_TIMEOUT_MINUTES`.

Migrating 3.x data to 4.x

The Continuous Delivery for PE 4.x series runs on a managed Kubernetes cluster, so your 3.x series data must be migrated to the new installation.

The migration process has four main steps:

1. Upgrade your existing 3.x installation to version 3.13.8.
2. Create a new installation of Continuous Delivery for PE version 4.x.
3. From the version 3.x root console, run the migration task.
4. Restart the 4.x installation, which is now populated with your 3.x data.

What gets migrated?

The migration task **copies** and migrates to 4.x:

- The database, which contains all your Continuous Delivery for PE 3.x installation's information, users, integrations, history, and artifacts except job commands and job logs.

  Note: If you've configured your 4.x installation to use an externally managed PostgreSQL database, this step is skipped.

- The object store, which contains all of your 3.x installation's job commands and job logs.

  Note: The 4.x series replaces external Artifactory and Amazon S3 object storage with a built-in highly available object storage system. Your 3.x object store will be migrated to the built-in 4.x object store.

- Any of the following configuration settings, if in use on 3.x:

```
CD4PE_REPO_CACHE_RETRIEVAL_TIMEOUT_MINUTES
CD4PE_LDAP_GROUP_SEARCH_SIZE_LIMIT
CD4PE_ENABLE_REPO_CACHING
CD4PE_HTTP_CONNECTION_TIMEOUT_SEC
CD4PE_HTTP_READ_TIMEOUT_SEC
CD4PE_HTTP_WRITE_TIMEOUT_SEC
CD4PE_HTTP_REQUEST_TIMEOUT_SEC
CD4PE_INCLUDE_GIT_HISTORY_FOR_CD4PE_JOBS
CD4PE_JOB_GLOBAL_TIMEOUT_MINUTES
CD4PE_JOB_HTTP_READ_TIMEOUT_MINUTES
```

The migration task **updates** in the 4.x installation:

- The webhooks between your source control system and the Continuous Delivery for PEbackend endpoint URL.

> **Note:** This step is optional when running the migration task. The migration task is designed to be safely run more than once, so you can make sure your 4.x installation is ready for full-time use before you enable webhook updates.

The migration task **does not copy** or move to 4.x:

- The 3.x root user, as you created a new root user when installing 4.x.
- The `JVM_ARGS` configuration setting, if in use on 3.x.
- Backup tables created for you by Continuous Delivery for PE during the 3.x series:
  - `app-pipelines-3-0-backup.pfi` - created during the upgrade from 2.x to 3.x
  - `pe-ia-node-results-cve-2020-7944-backup.pfi` and `pe-ia-resource-results-cve-2020-7944-backup.pfi` - created as part of the CVE 2020-7944 remediation in version 3.4.0

Migration cautions

A super user or the root user must perform this process.

Migrating from a local database in a 3.x installation to an external database in a 4.x installation is not supported.

New data created in the 3.x installation during the migration process might not be migrated to the 4.x installation. If data created during the migration process is missing after migration is complete, re-run the migration task.

The migration process is potentially memory-intensive, and can exceed the default Java heap size set by Continuous Delivery for PE. If you encounter an `out of memory` error during the migration process, double the heap size on your 4.x installation by adding Java virtual machine arguments such as the following on the **Config** page of the platform admin console:

```
-Xms512m -Xmx1G
```

Run the migration task

The migration task uses temporary credentials to establish a connection between your 3.x and 4.x installations, then migrates your 3.x installation data to a new 4.x installation.

Before you begin

1. **Upgrade your 3.x installation to version 3.13.8**. The migration task fails if you run it on any other 3.x version.

2. Create a new 4.x installation.

   a. Review the Puppet Application Manager system requirements.
   b. Follow the instructions that match your environment to Install Puppet Application Manager.
   c. According to your environment, either Configure and deploy Continuous Delivery for PE, or Configure and deploy Continuous Delivery for PE in an offline environment.

   CAUTION: Do not perform any integrations or create any new data in the 4.x installation until migration is complete.

3. Make sure that your 3.x installation can reach the Kubernetes API running on your 4.x installation.
4. If your new 4.x installation is behind a proxy, set both `HTTP_PROXY` and `HTTPS_PROXY` as values for the `cd4pe_docker_extra_params` parameter on the `cd4pe` class for your 3.x installation, passing in the URL of your proxy.

   Note: To pass additional arguments to the Docker process running the Continuous Delivery for PE container, you must specify them as an array. For example: `"-e HTTPS_PROXY=https://proxy.example.com", "-e HTTP_PROXY=https://proxy.example.com"`

1. In the platform admin console, click **Config**.
2. In the **Migration** area, select the **We're migrating an existing Continuous Delivery for PE 3.x instance** option. Scroll to the bottom of the page and click **Save config**.

   When this setting is enabled, data is allowed to move between your 3.x and 4.x installations.
3. When the success message appears, click **Go to new version**. On the **Version history** page, click **Deploy**.

4. Generate the temporary credentials needed to connect the two installations. On the server where you installed 4.x, run the following script:

```
#!/bin/bash
C_DEFAULT="\e[0m"
C_GREEN="\e[0;32m"

CD4PE_NAMESPACE="default"

K8S_SERVER_URL="$(kubectl config view --minify -o
 jsonpath='{.clusters[*].cluster.server}')"
K8S_TOKEN="$(kubectl -n $CD4PE_NAMESPACE get secret $(kubectl -n
 $CD4PE_NAMESPACE get secrets | grep ^cd4pe-migration | cut -f1 -d ' ') -o
 jsonpath="{['data']['token']}")"
if [ -z "${K8S_TOKEN}" ]; then
  printf "Kubernetes token not found.  Check to ensure 'Enable
 migration of an existing CD4PE instance' is checked in the application
 configuration\n"
  exit -1
fi

K8S_CA_CERT="$(kubectl config view --raw --minify -o
 jsonpath='{.clusters[*].cluster.certificate-authority-data}')"

printf "${C_GREEN}Kubernetes API server URL:${C_DEFAULT}\n\n%s\n\n" \
  "${K8S_SERVER_URL}"

printf "${C_GREEN}Kubernetes service account token:${C_DEFAULT}\n\n%s\n\n"
 \
  "${K8S_TOKEN}"

printf "${C_GREEN}Kubernetes API CA certificate:${C_DEFAULT}\n\n%s\n\n" \
  "${K8S_CA_CERT}"
```

> **Note:** If necessary, change the value of the `CD4PE_NAMESPACE` variable to your installation's namespace. Most installations use the `default` namespace.

5. In the Continuous Delivery for PE root console of your 3.x installation, navigate to **Settings** > **Migration**. Copy the credentials you generated in step 4 into the appropriate fields.
6. Click **Start migration** and confirm your settings.
7. Monitor the status of the migration task in the 3.x Docker logs. Access the logs by running `docker logs cd4pe`.

   The migration task first migrates your database tables, then your object store, and finally updates your webhooks, if applicable. When complete, a `Ran migration task in <time elapsed>` message is printed to the log.
8. Next, redeploy the 4.x installation. In the platform admin console, click **Config**.
9. In the **Migration** area, deselect the **We're migrating an existing Continuous Delivery for PE 3.x instance** option. Click **Save config**.
10. When the success message appears, click **Go to new version**. On the **Version history** page, locate the newly created version and click **Deploy**. Navigate to the 4.x installation, which now is populated with migrated data, and log in with your 3.x user credentials (not the root account).

Test the new 4.x installation

It's extremely important to thoroughly test the 4.x installation to make sure that the data has been properly migrated and all functionality is working as expected before moving on.

1. **Test the data migration.** Make sure that the data from your 3.x installation is now present in your 4.x installation by completing the following for each workspace in your 4.x installation:
   - Review the event logs for all control repos and modules
   - Check that all jobs and hardware connections are present
   - Check that all users and user groups are present
2. If you discover that any data is missing or looks incorrect, rerun the migration task.

> **Note:** Each time it runs, the migration task deletes any database tables it finds in the 4.x installation and recreates the tables from the 3.x installation's current status.

3. **Test manual pipeline runs.** To make sure that the 4.x installation is functioning properly, trigger a manual pipeline run for each control repo and module in each workspace in your 4.x installation.

   Run a pipeline manually by selecting **Trigger pipeline** from the **Manual actions** menu located above the pipeline.
4. When you've confirmed that all your 3.x data is present in the 4.x installation and manual pipeline runs complete successfully, move on to the next migration step: updating webhooks.

Update webhooks

As a final step in the 3.x to 4.x data migration process, or any time you change the location of your Continuous Delivery for PE installation, update the webhooks that connect Continuous Delivery for PE with your source control provider.

1. In the Continuous Delivery for PE 4.x root console, click **Settings** > **Webhooks**.
2. Enter the backend service endpoint of your previous Continuous Delivery for PE installation.

   If you're updating webhooks as the final step in a 3.x to 4.x data migration, enter the backend service endpoint shown on the **Settings** > **Endpoints** page in the 3.x root console.
3. Click **Update webhooks**.

   Continuous Delivery for PE updates all webhooks to point to the current installation.

Once you update your webhooks to point to the 4.x installation, do not create new data in the 3.x installation, and **do not run the migration task again**.

Post-migration next steps and troubleshooting

Once your 3.x data is migrated, the 4.x installation has been thoroughly tested, and webhooks are updated, you're ready to begin using 4.x with your team. However, you should not immediately decommission your 3.x installation, as it may be necessary to roll back to 3.x if issues arise.

Keep your 3.x installation intact until you are fully confident that all features of the new 4.x installation are working as expected. **As a precautionary measure, we strongly advise maintaining the inactive 3.x installation for the first two months of using 4.x.**

Maintaining the dormant 3.x installation for an extended test period allows you to roll back to 3.x and prevent production environment disruptions in the event that an issue arises with your 4.x migration.

Rolling back to the 3.x installation

If necessary, you can temporarily roll back to the 3.x installation by switching the active webhooks from the 4.x installation to the 3.x installation and generating new access tokens for all PE instances connected with the 3.x installation.

1. In your 3.x installation, navigate to a control repo's pipeline and click **Manage pipelines**.

2. Select **Manage in the web UI** and locate the **Automation webhook** section, where the webhook (including token) for your control repo is printed.
3. Copy the webhook and navigate to the corresponding repository in your source control provider. Add the webhook to the repository and remove the webhook that points to the 4.x installation.
4. Repeat this process for each control repo and module repo in your 3.x installation.
5. Generate a new access token for each PE instance connected with the 3.x installation by navigating to **Settings** > **Puppet Enterprise** and clicking **Edit credentials**  .
6. Select **Basic authorization** or **API token** and enter the required information:
   - For **Basic authorization**, enter the username and password for your "Continuous Delivery" user. Continuous Delivery for PE uses this information to generate an API token for you. The username and password are not saved.
   - For **API token**, generate a PE access token for your "Continuous Delivery" user using `puppet-access` or the RBAC v1 API, and paste it into the **API token** field. For instructions on generating an access token, see Token-based authentication.

     > **Note:** To avoid unintended service interruptions, create an access token with a multi-year lifespan.

Once the 3.x webhooks and new PE access tokens are in place, you can resume using the 3.x installation.

When you're ready to return to 4.x, repeat the migration process from the beginning.