



Puppet Enterprise 2021.0

Contents

Welcome to Puppet Enterprise® 2021.0.....	7
PE software architecture.....	7
Component versions in recent PE releases.....	13
FIPS 140-2 enabled PE.....	14
Getting support.....	15
Using the PE docs.....	23
Puppet platform documentation for PE.....	24
API index.....	26
 Release notes.....	 28
PE release notes.....	29
PE known issues.....	33
Platform release notes.....	38
Puppet release notes.....	39
Puppet known issues.....	49
Puppet Server release notes.....	50
Puppet Server known Issues.....	51
Facter release notes.....	52
Facter known issues.....	56
What's new since Puppet 6?.....	56
 Getting started with Puppet Enterprise.....	 57
Install PE.....	58
Add nodes to the inventory.....	59
Add code and set up Code Manager.....	61
Manage Apache configuration on *nix targets.....	65
Add a module.....	65
Configure your desired state.....	66
Organize webserver configurations with roles and profiles.....	67
Manage IIS configuration on Windows targets.....	70
Add a module.....	70
Configure your desired state.....	71
Organize webserver configurations with roles and profiles.....	72
Next steps.....	76
 Installing.....	 76
Supported architectures.....	77
System requirements.....	81
Hardware requirements.....	81
Supported operating systems.....	82
Supported browsers.....	87
System configuration.....	87
What gets installed and where?.....	97
Installing Puppet Enterprise.....	102
Purchasing and installing a license key.....	111
Installing agents.....	112

Installing compilers.....	132
Installing PE client tools.....	138
Uninstalling.....	142
Upgrading.....	144
Upgrading Puppet Enterprise.....	144
Upgrading agents.....	150
Configuring Puppet Enterprise.....	154
Tuning infrastructure nodes.....	154
Methods for configuring PE.....	160
Configuring Puppet Server.....	163
Configuring PuppetDB.....	166
Configuring security settings.....	167
Configuring proxies.....	169
Configuring the console.....	171
Configuring orchestration.....	173
Configuring ulimit.....	176
Writing configuration files.....	177
Analytics data collection.....	178
Static catalogs.....	182
Configuring disaster recovery.....	185
Disaster recovery.....	185
Configure disaster recovery.....	189
Accessing the console.....	194
Managing access.....	195
User permissions and user roles.....	196
Creating and managing local users and user roles.....	202
Connecting LDAP external directory services to PE.....	204
Working with user groups from a LDAP external directory.....	209
Connect a SAML identity provider to PE.....	210
Token-based authentication.....	218
RBAC API v1.....	223
Endpoints.....	224
Forming RBAC API requests.....	225
Users endpoints.....	226
User group endpoints.....	231
User roles endpoints.....	234
Permissions endpoints.....	237
Token endpoints.....	239
LDAP endpoints.....	242
SAML endpoints.....	244
Password endpoints.....	246
RBAC service errors.....	247
Configuration options.....	250
RBAC API v2.....	252
User group endpoints.....	252
Tokens endpoints.....	253

Activity service API.....	257
Forming activity service API requests.....	257
Event types reported by the activity service.....	258
Events endpoints.....	260

Monitoring and reporting..... 269

Monitoring infrastructure state.....	270
Viewing and managing packages.....	275
Value report.....	277
Infrastructure reports.....	281
Analyzing changes across Puppet runs.....	285
Viewing and managing Puppet Server metrics.....	287
Getting started with Graphite.....	288
Available Graphite metrics.....	292
Status API.....	296
Authenticating to the status API.....	297
Forming requests to the status API.....	297
JSON endpoints.....	298
Activity service plaintext endpoints.....	300
Metrics endpoints.....	301
The metrics API.....	307

Managing nodes..... 309

Adding and removing agent nodes.....	310
Adding and removing agentless nodes.....	312
How nodes are counted.....	315
Running Puppet on nodes.....	316
Grouping and classifying nodes.....	318
Making changes to node groups.....	326
Environment-based testing.....	328
Preconfigured node groups.....	330
Managing Windows nodes.....	334
Designing system configs: roles and profiles.....	360
The roles and profiles method.....	360
Roles and profiles example.....	364
Designing advanced profiles.....	367
Designing convenient roles.....	383
Node classifier API v1.....	387
Forming node classifier requests.....	388
Groups endpoint.....	389
Groups endpoint examples.....	407
Classes endpoint.....	409
Classification endpoint.....	411
Commands endpoint.....	421
Environments endpoint.....	422
Nodes check-in history endpoint.....	423
Group children endpoint.....	426
Rules endpoint.....	429
Import hierarchy endpoint.....	430
Last class update endpoint.....	431
Update classes endpoint.....	431
Validation endpoints.....	432
Node classifier errors.....	435
Node classifier API v2.....	436

Classification endpoint.....	437
Node inventory API.....	439
Managing patches.....	450
Configuring patch management.....	450
Patching nodes.....	456
Orchestrating Puppet runs, tasks, and plans.....	460
How Puppet orchestrator works.....	461
Setting up the orchestrator workflow.....	463
Configuring Puppet orchestrator.....	469
Running Puppet on demand.....	476
Running Puppet on demand from the console.....	476
Running Puppet on demand from the CLI.....	482
Running Puppet on demand with the API.....	486
Tasks in PE.....	489
Installing tasks.....	489
Running tasks in PE.....	490
Writing tasks.....	501
Plans in PE.....	517
Plans in PE versus Bolt plans.....	518
Installing plans.....	520
Running plans in PE.....	521
Writing plans.....	523
Puppet orchestrator API v1 endpoints.....	548
Puppet orchestrator API: forming requests.....	549
Puppet orchestrator API: command endpoint.....	550
Puppet orchestrator API: events endpoint.....	568
Puppet orchestrator API: inventory endpoint.....	574
Puppet orchestrator API: jobs endpoint.....	576
Puppet orchestrator API: scheduled jobs endpoint.....	587
Puppet orchestrator API: plans endpoint.....	593
Puppet orchestrator API: plan jobs endpoint.....	595
Puppet orchestrator API: tasks endpoint.....	603
Puppet orchestrator API: root endpoint.....	606
Puppet orchestrator API: usage endpoint.....	607
Puppet orchestrator API: scopes endpoint.....	608
Puppet orchestrator API: error responses.....	611
Migrating Bolt tasks and plans to PE.....	611
Managing and deploying Puppet code.....	614
Managing environments with a control repository.....	615
Managing environment content with a Puppetfile.....	620
Managing code with Code Manager.....	625
How Code Manager works.....	625
Configure Code Manager.....	626
Deploying Code without blocking requests to Puppet Server.....	630
Customize Code Manager configuration in Hiera.....	631
Triggering Code Manager on the command line.....	639
Triggering Code Manager with a webhook.....	645
Triggering Code Manager with custom scripts.....	646
Troubleshooting Code Manager.....	648
Code Manager API.....	651

Managing code with r10k.....	663
Configure r10k.....	664
Customizing r10k configuration.....	664
Deploying environments with r10k.....	671
r10k command reference.....	673
About file sync.....	674
SSL and certificates.....	677
Regenerate the console certificate.....	678
Regenerate infrastructure certificates.....	678
Use an independent intermediate certificate authority.....	681
Use a custom SSL certificate for the console.....	683
Change the hostname of a primary server.....	684
Generate a custom Diffie-Hellman parameter file.....	685
Enable TLSv1.....	686
Maintenance.....	686
Backing up and restoring Puppet Enterprise.....	686
Database maintenance.....	693
Troubleshooting.....	694
Log locations.....	695
Troubleshooting installation.....	696
Troubleshooting disaster recovery.....	697
Troubleshooting puppet infrastructure run commands.....	697
Troubleshooting connections between components.....	698
Troubleshooting the databases.....	700
Troubleshooting backup and restore.....	701
Troubleshooting Windows.....	701

Welcome to Puppet Enterprise® 2021.0

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

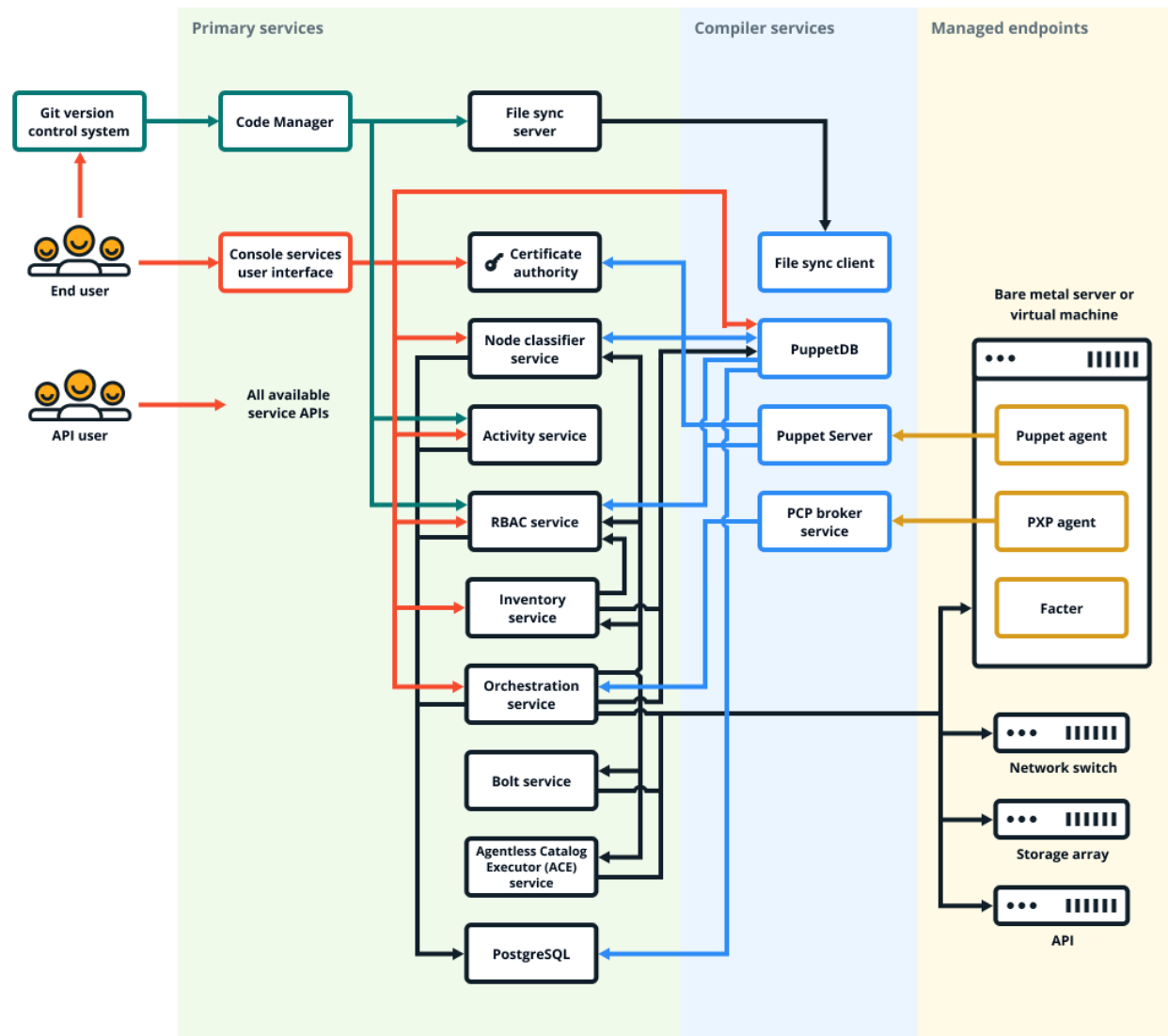
Puppet Enterprise docs links	Other useful links
Getting started Release notes Architecture overview System requirements Getting started guide	Docs for related Puppet products Open source Puppet Continuous Delivery for Puppet Enterprise Puppet Remediate Bolt Puppet Development Kit
Install and configure PE Install PE Install agents Add agentless nodes Configure and tune PE	Get help Support portal PE support lifecycle Archived PE docs
Manage your infrastructure Manage nodes Run jobs, tasks, and plans Deploy Puppet code	Share and contribute Puppet community Puppet Forge

To send us feedback or let us know about a docs error, [open a ticket](#) (you need a Jira account) or give the page a rating out of five stars and leave a comment.

PE software architecture

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

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



Related information

[Component versions in recent PE releases](#) on page 13

These tables show which components are in Puppet Enterprise (PE) releases, covering recent long-term supported (LTS) releases. To see component version tables for a release that has passed its support phase, switch to the docs site for that release.

The primary server and compilers

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

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

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

Puppet Server

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

Catalog compiler

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

File sync

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

Certificate Authority

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

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

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

Related information

[Hardware requirements](#) on page 81

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

[Installing compilers](#) on page 132

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

[About file sync](#) on page 674

File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

The Puppet agent

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

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

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

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

Puppet agent runs on *nix and Windows systems.

- [Puppet Agent on *nix Systems](#)
- [Puppet Agent on Windows Systems](#)

Facter

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

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

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

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

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

Console services

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

The console

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

The console can:

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

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

RBAC

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

For example:

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

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

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

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

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

Node classifier

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

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

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

You can also classify nodes using the NC API.

Related information

[Managing access](#) on page 195

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

[Managing nodes](#) on page 309

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

Code Manager and r10k

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

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

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

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

Related information

[Managing and deploying Puppet code](#) on page 614

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

[Triggering Code Manager on the command line](#) on page 639

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

[How Puppet orchestrator works](#) on page 461

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

Orchestration services

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

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

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

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

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

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

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

PE databases

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

The PE PostgreSQL instance includes the following databases:

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

PuppetDB

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

See the PuppetDB overview for more information.

Related information

[Database maintenance](#) on page 693

You can optimize the Puppet Enterprise (PE) databases to improve performance. For database maintenance, we recommend using the [pe_databases](#) module.

Security and communications

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

Service/Component	Communication Protocol	Authentication	Authorization
Puppet Server	HTTPS, SSL/TLS	SSL certificate verification with Puppet CA	trapperkeeper-auth
Certificate Authority	HTTPS, SSL/TLS	SSL certificate verification with Puppet CA	trapperkeeper-auth
Puppet agent	SSL/TLS	SSL certificate verification with Puppet CA	n/a
PuppetDB	HTTPS externally, or HTTP on the loopback interface	SSL certificate verification with Puppet CA	SSL certificate whitelist
PostgreSQL	PostgreSQL TCP, SSL for PE	SSL certificate verification with Puppet CA	SSL certificate whitelist

Service/Component	Communication Protocol	Authentication	Authorization
Activity service	SSL	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization
RBAC	SSL	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization
Classifier	SSL	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization
Console Services UI	SSL	Session-based authentication	RBAC user-based authorization
Orchestrator	HTTPS, Secure web sockets	RBAC token authentication	RBAC user-based authorization
PXP agent	Secure web sockets	SSL certificate verification with Puppet CA	n/a
PCP broker	Secure web sockets	SSL certificate verification with Puppet CA	trapperkeeper-auth
File sync	HTTPS, SSL/TLS	SSL certificate verification with Puppet CA	trapperkeeper-auth
Code Manager	HTTPS; can fetch code remotely via HTTP, HTTPS, and SSH (via Git)	RBAC token authentication; for remote module sources, HTTP(S) Basic or SSH keys	RBAC user-based authorization; for remote module sources, HTTP(S) Basic or SSH keys

Component versions in recent PE releases

These tables show which components are in Puppet Enterprise (PE) releases, covering recent long-term supported (LTS) releases. To see component version tables for a release that has passed its support phase, switch to the docs site for that release.

Puppet Enterprise agent and server components

This table shows the components installed on all agent nodes.

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

PE Version	Puppet and the Puppet agent	Facter	Hiera	Ruby	OpenSSL
2021.0	7.4.1	4.0.51	3.6.0	2.7.2	1.1.1i
2019.8.5 (LTS)	6.21.1	3.14.16	3.6.0	2.5.8	1.1.1i
2019.8.4	6.19.1	3.14.14	3.6.0	2.5.8	1.1.1g
2019.8.3	6.19.1	3.14.14	3.6.0	2.5.8	1.1.1g
2019.8.1	6.17.0	3.14.12	3.6.0	2.5.8	1.1.1g
2019.8	6.16.0	3.14.11	3.6.0	2.5.8	1.1.1g

This table shows components installed on server nodes.

PE Version	Puppet Server	PuppetDB	r10k	Bolt Services	Agentless Catalog Executor (ACE) Services	PostgreSQLJava		Nginx
2021.0	7.0.3	7.1.0	3.8.0	3.0.0	1.2.2	11.10	11.0	1.19.6
2019.8.5 (LTS)	6.15.1	6.14.0	3.8.0	3.0.0	1.2.2	11.10	11.0	1.19.6
2019.8.4	6.14.1	6.13.1	3.6.0	2.32.0	1.2.1	11.10	11.0	1.17.10
2019.8.3	6.14.1	6.13.1	3.6.0	2.32.0	1.2.1	11.9	11.0	1.17.10
2019.8.1	6.12.1	6.11.3	3.5.2	2.16.0	1.2.0	11.8	11.0	1.17.10
2019.8	6.12.0	6.11.1	3.5.1	2.11.1	1.2.0	11.8	11.0	1.17.10

Primary server and agent compatibility

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

	Server		
Agent	PE 2017.3 through 2018.1 Puppet 5.x	PE 2019.1 through 2019.8 Puppet 6.x	PE 2021.0 and later Puppet 7.x
5.x	#	#	
6.x	#	#	#
7.x	#	#	#

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

Task compatibility

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

PE version	Puppet task specification (GitHub)
2019.0.1+	version 1, revision 3
2019.0.0+	version 1, revision 2
2017.3.0+	version 1, revision 1

FIPS 140-2 enabled PE

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

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

Changes in FIPS-enabled PE installations

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

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

Limitations and cautions for FIPS-enabled PE installations

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

- Migrating from non-FIPS versions of PE to FIPS-enabled PE requires reinstalling on a [supported platform](#) with FIPS mode enabled.
- Disaster recovery configurations are not supported for FIPS-enabled PE.
- FIPS-enabled PE installations don't support extensions or modules that use the standard Ruby Open SSL library, such as `hiera-eyaml` or the `splunkhec` module. As a workaround, you can use a non-FIPS-enabled primary server with FIPS-enabled agents, which limits the issue to situations where only the agent uses the Ruby library.
- Due to a known issue with the `pe-client-tools` packages, `puppet code` and `puppet db` commands fail with SSL handshake errors when run on FIPS-compliant platforms. To use `puppet db` commands on a FIPS-compliant platform, install the [puppetdb_cli](#) Ruby gem. To use `puppet code` commands on a FIPS-compliant platform, use the [Code Manager API](#).

Related information

[Supported operating systems and devices](#) on page 82

When choosing an operating system, first consider the machine's role. Different roles support different operating systems and architectures.

Getting support

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

Puppet Enterprise support life cycle

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

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

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

Open source tools and libraries

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

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

Support portal

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

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

Note: We use the term "standard installation" to refer to a PE installation of up to 4,000 nodes. A Standard Support Plan is not limited to this installation type, however. Standard here refers to the support level, not the size of the PE installation.

Puppet metrics collector

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

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

The Puppet metrics collector is available from the Forge at https://forge.puppet.com/puppetlabs/puppet_metrics_collector.

PE support script

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

The script is provided by the `pe_support_script` module bundled with the installer.

Running the support script

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

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

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

The next iteration of the support script, version 3, is currently under development and may be selected by running `/opt/puppetlabs/bin/puppet enterprise support --v3`. This new version of the support script has more options, which can be used in addition to the list in the table above.

Option	Description
<code>--v3</code>	Activate version 3 of the support script. This option is required in order to use any other option listed in this table.
<code>--list</code>	List diagnostics that may be enabled or disabled. Diagnostics labeled "opt-in" must be explicitly enabled. All others are enabled by default.
<code>--enable <LIST></code>	A comma-separated list of diagnostic names to enable. Use the <code>--list</code> option to print available names. The <code>--enable</code> option must be used to activate diagnostics marked as "opt-in."
<code>--disable <LIST></code>	A comma-separated list of diagnostic names to enable. Use the <code>--list</code> option to print available names.
<code>--only <LIST></code>	A comma-separated list of diagnostic names to enable. All other diagnostics will be disabled. Use the <code>--list</code> option to print available names.
<code>--upload</code>	Upload the output tarball to Puppet Support via SFTP. Requires the <code>--ticket <NUMBER></code> option to be used.
<code>--upload_disable_host_key_check</code>	Disable SFTP host key checking. See Support article KB#0305 for a list of current host key values.
<code>--upload_user <USER></code>	Specify a SFTP user to use when uploading. If not specified, a shared write-only account will be used.
<code>--upload_key <FILE></code>	Specify a SFTP key to use with <code>--upload_user</code> .

Here are some examples of using the version 3 flags when running the support script:

```
# Collect diagnostics for just Puppet and Puppet Server
/opt/puppetlabs/bin/puppet enterprise support --v3 --only
puppet,puppetserver

# Enable collection of PE classification
/opt/puppetlabs/bin/puppet enterprise support --v3 --enable
pe.console.classifier-groups

# Disable collection of system logs, upload result to Puppet
/opt/puppetlabs/bin/puppet enterprise support --v3 --disable system.logs --
upload --ticket 12345
```

Descriptions of diagnostics that can be selected using the `--enable`, `--disable`, and `--only` flags are found in the next section.

Information collected by the support script

The following sections describe the information collected by version 1 and version 3 of the support script.

base-status

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

Information collected by the `base-status` check:

- The version of the support script that is running, the Puppet ticket number, if supplied, and the time at which the script was run.

system

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

Information collected by the `system.config` check:

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

Information collected by the `system.logs` check:

- A copy of the system log (`syslog`)
- A copy of the kernel log (`dmesg`)

Information collected by the `system.status` check:

- Values of variables set in the environment
- A list of running processes
- A list of enabled services
- The uptime of the system
- A list of established network connections
- NTP status
- The IP address and hostname of the node running the script, according to DNS
- Disk usage
- RAM usage

puppet

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

Information collected by the `puppet.config` check:

- Facter configuration files:
 - `/etc/puppetlabs/facter/facter.conf`
- Puppet configuration files:
 - `/etc/puppetlabs/puppet/device.conf`
 - `/etc/puppetlabs/puppet/hiera.yaml`
 - `/etc/puppetlabs/puppet/puppet.conf`
- PXP agent configuration files:
 - `/etc/puppetlabs/pxp-agent/modules/`
 - `/etc/puppetlabs/pxp-agent/pxp-agent.conf`

Information collected by the `puppet.logs` check:

- Puppet log files:
 - `/var/log/puppetlabs/puppet`
- JournalD logs for the puppet service

- PXP agent log files:
 - `/var/log/puppetlabs/puppet`
- JournalD logs for the `pxp-agent` service

Information collected by the `puppet.status` check:

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

puppetserver

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

Information collected by the `puppetserver.config` check:

- Puppet Server configuration files:
 - `/etc/puppetlabs/code/hiera.yaml`
 - `/etc/puppetlabs/puppet/auth.conf`
 - `/etc/puppetlabs/puppet/autosign.conf`
 - `/etc/puppetlabs/puppet/classfier.yaml`
 - `/etc/puppetlabs/puppet/fileserver.conf`
 - `/etc/puppetlabs/puppet/hiera.yaml`
 - `/etc/puppetlabs/puppet/puppet.conf`
 - `/etc/puppetlabs/puppet/puppetdb.conf`
 - `/etc/puppetlabs/puppet/routes.yaml`
 - `/etc/puppetlabs/puppetserver/bootstrap.cfg`
 - `/etc/puppetlabs/puppetserver/code-manager-request-logging.xml`
 - `/etc/puppetlabs/puppetserver/conf.d/`
 - `/etc/puppetlabs/puppetserver/logback.xml`
 - `/etc/puppetlabs/puppetserver/request-logging.xml`
 - `/etc/puppetlabs/r10k/r10k.yaml`
 - `/opt/puppetlabs/server/data/code-manager/r10k.yaml`

Information collected by the `puppetserver.logs` check:

- Puppet Server log files:
 - `/var/log/puppetlabs/puppetserver/`
- JournalD logs for the `pe-puppetserver` service
- `r10k` log files:
 - `/var/log/puppetlabs/r10k/`

Information collected by the `puppetserver.metrics` check:

- Data stored in `/opt/puppetlabs/puppet-metrics-collector/puppetserver`

Information collected by the `puppetserver.status` check:

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

puppetdb

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

Information collected by the `puppetdb.config` check:

- Configuration files:
 - `/etc/puppetlabs/puppetdb/bootstrap.cfg`
 - `/etc/puppetlabs/puppetdb/certificate-whitelist`
 - `/etc/puppetlabs/puppetdb/conf.d/`
 - `/etc/puppetlabs/puppetdb/logback.xml`
 - `/etc/puppetlabs/puppetdb/request-logging.xml`

Information collected by the `puppetdb.logs` check:

- PuppetDB log files: `/var/log/puppetlabs/puppetdb`
- JournalD logs for the `pe-puppetdb` service

Information collected by the `puppetdb.metrics` check:

- Data stored in `/opt/puppetlabs/puppet-metrics-collector/puppetdb`

Information collected by the `puppetdb.status` check:

- Output from the `status/v1/services` API
- Output from the `pdb/admin/v1/summary-stats` API
- A list of active certnames from the PQL query `nodes[certname] {deactivated is null and expired is null}`

pe

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

Information collected by the `pe.config` check:

- Installer configuration files:
 - `/etc/puppetlabs/enterprise/conf.d/`
 - `/etc/puppetlabs/enterprise/hiera.yaml`
 - `/etc/puppetlabs/installer/answers.install`
- PE client tools configuration files:
 - `/etc/puppetlabs/client-tools/orchestrator.conf`
 - `/etc/puppetlabs/client-tools/puppet-access.conf`
 - `/etc/puppetlabs/client-tools/puppet-code.conf`
 - `/etc/puppetlabs/client-tools/puppetdb.conf`
 - `/etc/puppetlabs/client-tools/services.conf`

Information collected by the `pe.logs` check:

- PE installer log files:
 - `/var/log/puppetlabs/installer/`
- PE backup and restore log files:
 - `/var/log/puppetlabs/pe-backup-tools/`
 - `/var/log/puppetlabs/puppet_infra_recover_config_cron.log`

Information collected by the `pe.status` check:

- Output from `puppet infra status`
- Current tuning settings from `puppet infra tune`
- Recommended tuning settings from `puppet infra tune`

This check is disabled by default. Information collected by the `pe.file-sync` check when activated by the `--enable` option:

- Puppet manifests and other content from `/etc/puppetlabs/code-staging/`
- Puppet manifests and other content stored in Git repos under `/opt/puppetlabs/server/data/puppetserver/filesync`

pe.console

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

Information collected by the `pe.console.config` check:

- Configuration files:
 - `/etc/puppetlabs/console-services/bootstrap.cfg`
 - `/etc/puppetlabs/console-services/conf.d/`
 - `/etc/puppetlabs/console-services/logback.xml`
 - `/etc/puppetlabs/console-services/rbac-certificate-whitelist`
 - `/etc/puppetlabs/console-services/request-logging.xml`
 - `/etc/puppetlabs/nginx/conf.d/`
 - `/etc/puppetlabs/nginx/nginx.conf`

Information collected by the `pe.console.logs` check:

- Console log files:
 - `/var/log/puppetlabs/console-services/`
 - `/var/log/puppetlabs/nginx/`
- JournalD logs for the `pe-puppetdb` and `pe-nginx` services

Information collected by the `pe.console.status` check:

- Output from the `/status/v1/services` API
- Directory service connection configuration, with passwords removed

This check is disabled by default. Information collected by the `pe.console.classifier-groups` check when activated by the `--enable` option:

- All classification data provided by the `/v1/groups` API endpoint

pe.orchestration

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

Information collected by the `pe.orchestration.config` check:

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

Information collected by the `pe.orchestration.logs` check:

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

Information collected by the `pe.orchestration.metrics` check:

- Data stored in `/opt/puppetlabs/puppet-metrics-collector/orchestrator/`

Information collected by the `pe.orchestration.status` check:

- Output from the `/status/v1/services` API

pe.postgres

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

Information collected by the `pe.postgres.config` check:

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

Information collected by the `pe.postgres.logs` check:

- PostgreSQL log files:
 - `/var/log/puppetlabs/postgresql/*/`
 - `/opt/puppetlabs/server/data/postgresql/pg_upgrade_internal.log`
 - `/opt/puppetlabs/server/data/postgresql/pg_upgrade_server.log`
 - `/opt/puppetlabs/server/data/postgresql/pg_upgrade_utility.log`
- JournalD logs for the `pe-postgresql` service

Information collected by the `pe.postgres.status` check:

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

Community support

As a Puppet Enterprise customer you are more than welcome to participate in our large and helpful open source community as well as report issues against the open source project.

- Join the [Puppet Enterprise user group](#). Your request to join is sent to Puppet, Inc. for authorization and you receive an email when you've been added to the user group.
 - Click on "Sign in and apply for membership."
 - Click on "Enter your email address to access the document."
 - Enter your email address.
- Join the open source [Puppet user group](#).
- Join the [Puppet developers group](#).
- Report issues with the [open source Puppet project](#).

Using the PE docs

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

Archived PE docs

PE docs for recent end-of-life (EOL) or superseded product versions are archived in place, meaning that we continue to host them at their original URLs, but we limit their visibility on the main docs site and no longer update them. You can access archived-in-place docs using their original URLs, or from the links here.

PE docs for EOL versions earlier than those listed here are archived in our [PE docs GitHub archive](#).

PE Version	URL
2019.7	https://puppet.com/docs/pe/2019.7/pe_user_guide.html
2019.5	https://puppet.com/docs/pe/2019.5/pe_user_guide.html
2019.4	https://puppet.com/docs/pe/2019.4/pe_user_guide.html
2019.3	https://puppet.com/docs/pe/2019.3/pe_user_guide.html
2019.2.z	https://puppet.com/docs/pe/2019.2/pe_user_guide.html
2019.1.z	https://puppet.com/docs/pe/2019.1/pe_user_guide.html
2019.0.z	https://puppet.com/docs/pe/2019.0/pe_user_guide.html
2018.1.z	https://puppet.com/docs/pe/2018.1/pe_user_guide.html
2017.3.z	https://puppet.com/docs/pe/2017.3/pe_user_guide.html
2017.2.z	https://puppet.com/docs/pe/2017.2/index.html
2017.1.z	https://puppet.com/docs/pe/2017.1/index.html

PE Version	URL
2016.5.z	https://puppet.com/docs/pe/2016.5/index.html
2016.4.z	https://puppet.com/docs/pe/2016.4/index.html

Usage notes for example commands

Review these guidelines to help you understand and use example commands throughout PE docs.

Using puppet commands to generate cURL arguments

The examples used throughout the PE docs rely on puppet commands to populate some cURL arguments, taking the guesswork out of providing those values. For example, you might see something like this:

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

Puppet commands can return different values depending on various conditions. In order for the cURL examples to work as intended, run the entire example (including setting the environment variables and calling the curl command) as root, Administrator, or with equivalent elevated privileges.

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

Alternative ways to include authentication tokens

Any curl example that contains this line:

```
auth_header="X-Authentication: $(puppet-access show)"
```

can instead use an actual token if you have one available:

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

Puppet platform documentation for PE

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

Puppet

- [Puppet docs](#)

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

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

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

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

Language

- [An outline of how the Puppet language works.](#)
- [Resources](#), [variables](#), [conditional statements](#), and [relationships and ordering](#) are the fundamental pieces of the Puppet language.

- [Classes](#) and [defined types](#) are how you organize Puppet code into useful chunks. Classes are the main unit of Puppet code you'll be interacting with on a daily basis. You can assign classes to nodes in the PE console.
- [Facts and built-in variables](#) explains the special variables you can use in your Puppet manifests.

Modules

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

Services and commands

- [An overview of Puppet's architecture](#).
- [A list of the main services and commands you'll interact with](#).
- [Notes on running Puppet's commands on Windows](#).

Built-in resource types and functions

- [The resource type reference](#) has info about all of the built-in Puppet resource types.
- [The function reference](#) does the same for the built-in functions.

Important directories and files

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

Configuration

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

Puppet Server

- [Puppet Server docs](#)

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

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

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

Factor

- [Factor docs](#)

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

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

Hiera

- [Hiera docs](#)

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

Start with the [overview](#) and use the navigation sidebar to get around.

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

PuppetDB

- [PuppetDB docs](#)

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

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

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

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

Related information

[Managing and deploying Puppet code](#) on page 614

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

API index

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

Puppet Enterprise APIs

For information on port requirements, see the [System Configuration](#) documents.

API	Useful for
Node inventory API on page 439	<ul style="list-style-type: none"> • Making HTTP(S) requests to the Puppet inventory service API. • Creating and deleting connection entries in the inventory service database. • Listing the connections entries in the inventory database.

API	Useful for
RBAC service API v1	<ul style="list-style-type: none"> Managing access to Puppet Enterprise. Connecting to external directories. Generating authentication tokens. Managing users, user roles, user groups, and user permissions.
RBAC service API v2	<ul style="list-style-type: none"> Revoking authentication tokens.
Node classifier service API	<ul style="list-style-type: none"> Querying the groups that a node matches. Querying the classes, parameters, and variables that have been assigned to a node or group. Querying the environment that a node is in.
Orchestrator API	<ul style="list-style-type: none"> Gathering details about the orchestrator jobs you run. Inspecting applications and applications instances in your Puppet environments.
Code Manager API	<ul style="list-style-type: none"> Creating a webhook to trigger Code Manager. Queueing Puppet code deployments. Checking Code Manager and file sync status.
Status API	<ul style="list-style-type: none"> Checking the health status of PE services.
Activity service API	<ul style="list-style-type: none"> Querying PE service and user events logged by the activity service.
Value API	<ul style="list-style-type: none"> Generating reports about time and money freed by PE automation.

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

API	Useful for
Puppet Server administrative API endpoints <ul style="list-style-type: none"> environment-cache jruby-pool 	<ul style="list-style-type: none"> Deleting environment caches created by a primary server. Deleting the Puppet Server pool of JRuby instances.

API	Useful for
Server-specific Puppet API <ul style="list-style-type: none"> • Environment classes • Environment modules • Static file content 	<ul style="list-style-type: none"> • Getting the classes and parameter information that is associated with an environment, with cache support. • Getting information about what modules are installed in an environment. • Getting the contents of a specific version of a file in a specific environment.
Puppet Server status API	<ul style="list-style-type: none"> • Checking the state, memory usage, and uptime of the services running on Puppet Server.
Puppet Server metrics API <ul style="list-style-type: none"> • v1 metrics (deprecated) • v2 metrics (Jolokia) 	<ul style="list-style-type: none"> • Querying Puppet Server performance and usage metrics.
Puppet HTTP API	<ul style="list-style-type: none"> • Retrieving a catalog for a node • Accessing environment information
Certificate Authority (CA) API	<ul style="list-style-type: none"> • Used internally by Puppet to manage agent certificates.
PuppetDB APIs	<ul style="list-style-type: none"> • Querying the data that PuppetDB collects from Puppet • Importing and exporting PuppetDB archives • Changing the PuppetDB model of a population • Querying information about the PuppetDB server • Querying PuppetDB metrics
Forge API	<ul style="list-style-type: none"> • Finding information about modules and users on the Forge • Writing scripts and tools that interact with the Forge website

Release notes

These release notes contain important information about Puppet Enterprise® 2021.0.

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

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

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

- [PE release notes](#) on page 29

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

- [PE known issues](#) on page 33

These are the known issues in PE 2021.0.

- [Platform release notes](#) on page 38

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

PE release notes

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

PE 2021.0

Released February 2021

New features

SAML support

SAML 2.0 support allows you to securely authenticate users with single sign-on (SSO) and/or multi-factor authentication (MFA) through your SAML identity provider. To configure SAML in the console, see [Connect a SAML identity provider to PE](#) on page 210.

Enhancements

Generate, view, and revoke tokens in the console

In the console, on the **My account** page, in the **Tokens** tab, you can create and revoke tokens, or view a list of your currently active tokens. Administrators can view and revoke another user's tokens on the **User details** page.

Migrate CA files to the new default directory

The default CA directory has moved to a new location at `/etc/puppetlabs/puppetserver/ca` from its previous location at `/etc/puppetlabs/puppet/ssl/ca`. This change helps prevent unintentionally deleting your CA files in the process of regenerating certificates. If applicable, you're prompted with CLI instructions for migrating your CA directory after upgrade.

```
/opt/puppetlabs/bin/puppet resource service pe-puppetserver ensure=stopped
/opt/puppetlabs/bin/puppetserver ca migrate
/opt/puppetlabs/bin/puppet resource service pe-puppetserver ensure=running
/opt/puppetlabs/bin/puppet agent -t
```

Install the Puppet agent despite issues in other yum repositories

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

Get better insight into replica sync status after upgrade

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

Fix replica enablement issues

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

Patch nodes with built-in health checks

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

Run a command after patching nodes

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

Patch nodes despite certain read-only directory permissions

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

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

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

Use Hiera lookups outside of apply blocks in plans

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

See the duration of Puppet and plan runs

New `duration`, `created_timestamp`, and `finished_timestamp` keys allow you to see the duration of Puppet and plan runs in the [GET /jobs](#) and [GET /plan_jobs](#) on page 595 endpoints.

View the error location in plan error details

The puppet plan functions provide the file and line number where the error occurred in the `details` key of the error response.

Run plans on PuppetDB queries and node classifier group targets

The `params` key in the [POST /command/environment_plan_run](#) on page 566 endpoint allows you to specify PuppetDB queries and node groups as targets during a plan run.

Use masked inputs for sensitive parameters

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

Configure how many times the orchestrator allows status request timeouts

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

Accept and store arbitrary data related to a job

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

- POST /command/deploy
- POST /command/task
- POST /command/plan_run
- POST /command/environment_plan_run

The key is returned in the following endpoints:

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

Sort and reorder nodes in node lists

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

Add custom parameters when installing agents in the console

In the console, on the **Install agent on nodes**, you can click **Advanced install** and add custom parameters to the `pe_bootstrap` task during installation.

Update facts cache terminus to use JSON

The facts cache terminus is now JSON by default. To switch back to YAML, set `puppet_enterprise::profile::master::puppetdb::facts_cache_terminus` to `yaml` in `hieradata`.

Configure failed deployments to display r10k stacktrace in error output

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

Reduce query time when querying nodes with a fact filter

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

Deprecations and removals

Platforms deprecated

Support for these agent platforms is deprecated in this release.

- Enterprise Linux 5

- Enterprise Linux 7 ppc64le
- SUSE Linux Enterprise Server 11
- SUSE Linux Enterprise Server 12 ppc64le
- Ubuntu 16.04 ppc64le
- Debian 8
- Solaris 10
- Microsoft Windows 7, 8.1
- Microsoft Windows Server 2008, 2008 R2

Platforms removed

Support for these agent platforms is removed in this release. Before upgrading to this version, remove the `pe_repo::platform` class for these operating systems from the **PE Master** node group in the console, and from your code and Hiera.

- AIX 6.1
- Enterprise Linux 4
- Enterprise Linux 6, 7 s390x
- Fedora 26, 27, 28, 29
- Mac OS X 10.9, 10.12, 10.13
- SUSE Linux Enterprise Server 11, 12 s390x

Resolved issues

PuppetDB restarted continually after upgrade with deprecated parameters

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

Tasks failed when specifying both as the input method

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

Patch task misreported success when it timed out on Windows nodes

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

Orchestrator created an extra JRuby pool

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

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

The command provided in the console to install Windows nodes installed a non-FIPS agent regardless of the node's FIPS status.

Unfinished sync reported as finished when clients shared the same identifier

Because the orchestrator and puppetserver file sync clients shared the same identifier, Code Manager reported an unfinished sync as `"all-synced": true`. Whichever client finished polling first, notified the storage service that

the sync was complete, regardless of the other client's sync status. This reported sync might have caused attempts to access tasks and plans before the newly-deployed code was available.

Refused connection in orchestrator startup caused PuppetDB migration failure

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

Upgrade failed with Hiera data based on certificate extensions

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

File sync issued an alert when a repository had no commits

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

Upgrade failed with infrastructure nodes classified based on trusted facts

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

Patch task failed on Windows nodes with old logs

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

Backups failed if a Puppet run was in progress

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

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

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

Module-only environment updates did not deploy in Versioned Deploys

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

PE known issues

These are the known issues in PE 2021.0.

Installation and upgrade known issues

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

Compiler upgrade fails with no-op configured

Upgrade fails on compilers running in no-op mode. As a workaround, remove the no-op setting on compilers before upgrading.

Compiler upgrade fails with client certnames defined

Existing settings for client certnames can cause upgrade to fail on compilers, typically with the error `Value does not match schema: {:client-certnames disallowed-key}`. As a workaround, manually remove any `client-certnames` settings for compilers from `/etc/puppetlabs/puppetserver/conf.d/file-sync.conf`.

Windows agent installation fails with a manually transferred certificate

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

Initial agent run after upgrade can fail with many environments

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

Installing Windows agents with the .msi package fails with a non-default INSTALLEDIR

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

Upgrade fails with cryptic errors if agent_versions are configured for your infrastructure pe_repo class

If you've configured the `agent_version` parameter for the `pe_repo` class that matches your infrastructure nodes, upgrade can fail with a timeout error when the installer attempts to download a non-default agent version. As a workaround, before upgrading, remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters for the `pe_repo` class that matches your infrastructure nodes.

Converting legacy compilers fails with an external certificate authority

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

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

To work around this issue when it appears:

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

Converted compilers can slow PuppetDB in geo-diverse installations

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

Upgrading to newer version with versioned deploys causes Puppet Server to crash

If `versioned_deploys` is enabled when upgrading to version 2019.8.6 or 2021.1, then the Puppet Server crashes.

Disaster recovery known issues

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

Puppet runs can take longer than expected in failover scenarios

In disaster recovery environments with a provisioned replica, if the primary server is unreachable, a Puppet run using data from the replica can take up to three times as long as expected (for example, 6 minutes versus 2 minutes).

FIPS known issues

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

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

FIPS-enabled PE installations don't support extensions or modules that use the standard Ruby Open SSL library, such as `hier-eyaml` or the `splunkhec` module. As a workaround, you can use a non-FIPS-enabled primary server with FIPS-enabled agents, which limits the issue to situations where only the agent uses the Ruby library.

Errors when using `puppet code` and `puppet db` commands on FIPS-compliant platforms

When the `pe-client-tools` packages are run on FIPS-compliant platforms, `puppet code` and `puppet db` commands fail with SSL handshake errors. To use `puppet db` commands on a FIPS-compliant platform, install the [puppetdb_cli](#) Ruby gem with the following command:

```
/opt/puppetlabs/puppet/bin/gem install puppetdb_cli --bindir /opt/puppetlabs/bin/
```

To use `puppet code` commands on a FIPS-compliant platform, use the Code Manager API. Alternatively, you can use `pe-client-tools` on a non-FIPS-compliant platform to access a FIPS-enabled primary server.

Configuration and maintenance known issues

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

Restarting or running Puppet on infrastructure nodes can trigger an `illegal reflective access operation warning`

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

Backup fails with an error about the `stockpile` directory

The `puppet-backup create` command fails under certain conditions with an error that the `/opt/puppetlabs/server/data/puppetdb/stockpile` directory is inaccessible.

Orchestration services known issues

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

Running plans during code deployment can result in failures

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

Pantomime dependency in the orchestrator

The version of pantomime in the orchestrator had a third party vulnerability (tika-core). Because of the vulnerability, pantomime usage was removed from the orchestrator, but pantomime still exists in the orchestration-services build.

Console and console services known issues

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

Console reboot task fails

Rebooting a node using the reboot task in the console fails due to the removal of win32 gems in Puppet 7. As a workaround, update the reboot module to version 4.0.2 or later.

Gateway timeout errors in the console

Using facts to filter nodes might produce either a "502 Bad Gateway" or "Gateway Timeout" error instead of the expected results.

Injection attack vulnerability in csv exports

There's a vulnerability in the console where .csv files could contain malicious user input when exported.

Patching known issues

These are the known issues for patching in this release.

Patching fails on Windows nodes when run during a fact generation

The patching task and plan fail on Windows nodes if run during fact generation. Because both processes use the same lock file, you see an exception about a duplicate lock file, for example:

```
Add-LogEntry : Exception caught: Lock file found, it appears PID is another
copy of pe_patch_fact_generation or pe_patch_groups
```

As a workaround, re-run the patching task or plan when fact generation completes.

Patching fails on Windows nodes with non-default agent location

On Windows nodes, if the Puppet agent is installed to a location other than the default C: drive, the patching task or plan fails with the error No such file or directory.

Patching fails with excluded yum packages

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

Code management known issues

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

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

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

Enabling Code Manager and multithreading in Puppet Server deadlocks JRuby

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

Default SSH URL with TFS fails with Rugged error

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

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

For example, change:

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

to

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

GitHub security updates might cause errors with `shellgit`

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

Timeouts when using `--wait` with large deployments or geographically dispersed compilers

Because the `--wait` flag deploys code to all compilers before returning results, some deployments with a large node count or compilers spread across a large geographic area might experience a timeout. Work around this issue by adjusting the `timeouts_sync` parameter.

r10k with the Rugged provider can develop a bloated cache

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

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

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

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

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

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

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

Repeat these steps on each primary server exhibiting the same symptoms, including any compilers.

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

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

The following error appears in `puppetserver.log`:

```
2016-05-05 11:57:06,042 ERROR [clojure-agent-send-off-pool-0] [p.e.s.f.file-sync-client-core] Fatal error during file sync, requesting shutdown.
org.eclipse.jgit.api.errors.JGitInternalException: Could not delete file /
etc/puppetlabs/code/environments/development
    at org.eclipse.jgit.api.CleanCommand.call(CleanCommand.java:138)
~[puppet-server-release.jar:na]
```

To recover from this error:

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

Code Manager can't recover from Puppetfile typos in URL

When you have a git typo in your Puppetfile, subsequent code deploys continuously fail until you manually delete deployer caches, even after the Puppetfile error is corrected.

File sync fails to copy symlinks if versioned deploys is enabled

If you enable versioned deploys, then the file sync fails to copy symlinks and incorrectly copies the symlinks' targets instead. This copy failure crashes the Puppet Server.

Platform release notes

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

Open source component	Version used in PE
Puppet and the Puppet agent	7.4.1
Puppet Server	7.0.3
PuppetDB	7.1.0
Facter	4.0.51

- [Puppet release notes](#) on page 39

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

- [Puppet known issues](#) on page 49

These are the known issues in this version of Puppet.

- [Puppet Server release notes](#) on page 50

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

- [Puppet Server known Issues](#) on page 51

These are the known issues in this version of Puppet Server.

- [PuppetDB release notes \(link\)](#)

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

- [Factor release notes](#) on page 52

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

- [Factor known issues](#) on page 56

These are the known issues in this version of Factor.

- [What's new since Puppet 6?](#) on page 56

These are the major new features, enhancements, deprecations, and removals since the Puppet 6 release.

Puppet release notes

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

Puppet 7.4.1

Released 16 February 2021.

Resolved issues

Puppet users with `forcelocal` are no longer idempotent

This release fixes a regression where setting the `gid` parameter on a user resource with `forcelocal` was not idempotent. [PUP-10896](#)

Puppet 7.4.0

Released 9 February 2021.

New features

`--timing` option in `puppet facts show`

This release adds a `--timing` option in the `puppet facts show` command. This flag shows you how much time it takes to resolve each fact. [PUP-10858](#)

Resolved issues

User resource with `forcelocal` uses `getent` for groups

The `useradd` provider now checks the `forcelocal` parameter and gets local information on the groups (from `/etc/groups`) and `gid` (from `etc/passwd`) of the user when requested. [PUP-10857](#)

Slow Puppet agent run after upgrade to version 6

This release improves the performance of the `apt` package provider when removing packages by reducing the calls to `apt-mark showmanual`. [PUP-10856](#)

The `apt` provider does not work with local packages

The `apt` package provider now allows you to install packages from a local file using `source` parameter. [PUP-10854](#)

The `puppet facts show --value-only` command displays a quoted value

Previously, the `puppet facts show --value-only <fact>` command emitted the value as a JSON string, which included quotes around the value, such as `{ "RedHat" }`. It now only emits the value. [PUP-10861](#)

Puppet 7.3.0

Released 20 January 2021.

New features

New `serverport` setting type

The `serverport` setting is an alias for `masterport`. [PUP-10725](#)

Enhancements

Multiple `logdest` locations in `puppet.conf` accepted

You can set multiple `logdest` locations using a comma separated list. For example: `/path/file1,console,/path/file2`. [PUP-10795](#)

The `puppet module install` command lists unsatisfiable dependencies

If the `puppet module install` command fails, Puppet returns a more detailed error, including the unsatisfiable module(s) and its ranges. [PUP-9176](#)

New `--no-legacy` option to disable legacy facts

By default, `puppet facts show` displays all facts, including legacy facts. This release adds a `--no-legacy` option to disable legacy facts when querying all facts. [PUP-10850](#)

Resolved issues

The `puppet apply` command creates warnings

This release eliminates Ruby 2.7.x warnings when running `puppet apply` with node statements. [PUP-10845](#)

Remove `Pathname#cleanpath` workaround

This release removes an unnecessary workaround when cleaning file paths, as Ruby 1.9 is no longer supported. [PUP-10840](#)

The `allow *` error message shown during PE upgrade

Puppet no longer prints an error if `fileserver.conf` contains `allow *` rules. It continues to print an error for all other rules, as Puppet's legacy authorization is no longer supported and is superseded by Puppetserver's authorization. [PUP-10851](#)

3x functions cannot be called from deferred functions in Puppet agent

This release allows deferred 3.x functions, like `sprintf`, to be called during a Puppet agent run. [PUP-10819](#)

Cached catalog contains the result of deferred evaluation instead of the deferred function

Puppet 6.12.0 introduced a regression that caused the result of a deferred function to be stored in the cached catalog. As a result, an agent running with a cached catalog would not re-evaluate the deferred function. This is now fixed. [PUP-10818](#)

puppet facts show fact output differs from facter fact

The output format is different between Facter and Puppet facts when a query for a single fact is provided. This is now fixed. [PUP-10847](#)

Issue with Puppet creating production folder when multiple environment paths are set

Previously, the `production` environment folder was automatically created at every Puppet ran in the first search path, if it did not already exist. This release ensures Puppet searches all the given paths before creating a new `production` environment folder. [PUP-10842](#)

Puppet 7.2.0

This version of Puppet was never released.

Puppet 7.1.0

Released 15 December 2020.

Enhancements**Reduced query time for system user groups**

The time it takes to query groups of a system user has been reduced on Linux operating systems with FFI. The `getgrouplist` method is also available. [PUP-10774](#)

Log rotation for Windows based platforms

You can now configure the `pxp-agent` to use the Windows Event Log service by setting the `logfile` value to `eventlog`. [PA-3492](#)

Log rotation for macOS based platforms

This release enables log rotation for the `pxp-agent` on OSX platforms. [PA-3491](#)

Added server alias for routes.yaml

When `routes.yaml` is parsed, it accepts either `server` or `master` applications. [PUP-10773](#)

OpenSSL bumped to 1.1.1i

This release bumps OpenSSL to 1.1.1i. [PA-3513](#)

Curl bumped to 7.74.0

This release bumps Curl to 7.74.0. [PA-3512](#)

Resolved issues**The Puppet 7 gem is missing runtime dependency on scanf**

This is fixed and you can now run module tests against the Puppet gem on Ruby 2.7. [PUP-10797](#)

The puppet node clean action LoggerIO needs to implement warn

In Puppet 7.0.0, the `puppet node clean` action failed if you had `cadir` in the legacy location or inside the `ssldir`. This was a regression and is now fixed. [PUP-10786](#)

Calling scope#tags results in undefined method

Previously, calling the `tags` method within an ERB template resulted in a confusing error message. The error message now makes it clear that this method is not supported. [PUP-10779](#)

User resource is not idempotent on AIX

The AIX user resource now allows for password lines with arbitrary whitespace in the `passwd` file. [PUP-10778](#)

Fine grained environment timeout issues

Previously, if the `environment.conf` for an environment was updated and the environment was cleared, `puppetserver` used old values for per-environment settings. This happened if the environment timed out or if the environment was explicitly cleared using `puppetserver`'s environment cache REST API. With this fix, if an environment is cleared, Puppet reloads the per-environment settings from the updated `environment.conf`. [PUP-10713](#)

FIPS compliant nodes are returning an error

This release fixes an issue on Windows FIPS where Leatherman libraries loaded at the predefined address of the OpenSSL library. This caused the OpenSSL library to relocate to a different address, failing the FIPS validation. This is fixed and leatherman compiled with `dynamicbase` is disabled on Windows. [PA-3474](#)

User provider with uid/gid as Integer raises warning

This release fixes a warning introduced in Ruby 2.7 that checked invalid objects (such as Integer) against a regular expression. [PUP-10790](#)

Puppet 7.0.0

Released 19 November 2020.

For a list of major changes, see [What's new since Puppet 7](#).

New features

The `puppet facts show` command

You can use the `puppet facts show` command to retrieve a list of facts. By default, it does not return legacy facts, but you can enable it to with the `--show legacy` option. This command replaces `puppet facts find` as the default Puppet facts action. [PUP-10644](#) and [PUP-10715](#)

JSON terminus for node and report

This release implements JSON termini for node and report indirection. The format of the `last_run_report.yaml` report can be affected by the `cache` setting key of the `report terminus` in the `routes.yaml` file. To ensure the file extension matches the content, update the `lastrunreport` configuration to reflect the terminus changes (`lastrunreport = $statedir/last_run_report.json`). [PUP-10712](#)

JSON terminus for facts

This release adds a new JSON terminus for facts, allowing them to be stored and loaded as JSON. Puppet agents continue to default to YAML, but you can use JSON by configuring the agent application in `routes.yaml`. Puppet Server 7 also caches facts as JSON instead of YAML by default. You can re-enable the old YAML terminus in `routes.yaml`. [PUP-10656](#)

Public folder

There is a new folder with 0755 access rights named `public`, which is now the default location for the `last_run_summary.yaml` report. It has 640 file permissions. This makes it possible for a non-privileged process to read the file. To relax permissions on the last run summary, set the group permission on the file in `puppet.conf` to the following [PUP-10627](#): `lastrunsummary = $publicdir/last_run_summary.yaml { owner = root, group = monitoring, mode = 0640 }`

The `settings_catalog` setting

To load Puppet more quickly, you can set the `settings_catalog` setting to `false` to skip applying the settings catalog. The setting defaults to `true`. [PUP-8682](#)

New numeric and port setting types

This release adds a new `port` setting type, which turns the given value to an integer, and validates it if the value is in the range of 0-65535. Puppet port can use this setting type. [PUP-10711](#)

MSI `PUPPET_SERVER` and alias

This release adds a new Windows Installer property called `PUPPET_SERVER`. You can use this as an alias to the existing `PUPPET_MASTER_SERVER` property. [PA-3440](#)

New GPG signing key

Puppet has a new GPG signing key. See [verify packages](#) for the new key.

Enhancements

Ruby version bumped to 2.7

The default version of Ruby is now 2.7. The minimum Ruby version required to run Puppet 7 is now 2.5. After upgrading to Puppet 7, you may need to use the `puppet_gem` provider to ensure all your gems are installed. [PUP-10625](#)

Default digest algorithm changed to sha256

Puppet 7 now uses sha256 as the default digest algorithm. [PUP-10583](#)

Gem provider installs gems in Ruby

The gem provider now installs gems in Ruby by default. Use the `puppet_gem` provider to reinstall gems in the Ruby distribution vendored in Puppet. For example, if custom providers or deferred functions require gems during catalog application. [PUP-10677](#)

FFI functions, structs and constants moved to a separate Windows module

To increase speed, we have moved FFI functions, constants and structures out of `Puppet::Util::Windows`. [PUP-10606](#)

Default value of `ignore_plugin_errors` changed from `true` to `false`

The default value for `ignore_plugin_errors` is now `false`. This stops Puppet agents failing to `pluginsync`. [PUP-10598](#)

Interpolation of sensitive values in EPP templates

Previously, if you interpolated a sensitive value in a template, you were required to unwrap the sensitive value and rewrap the result. Now the `epp` and `inline_epp` functions automatically return a Sensitive value if any interpolated variables are sensitive. For example: `inline_epp("Password is <%= Sensitive('opensesame') %>")`. Note that these changes just apply to EPP templates, not ERB templates. [PUP-8969](#)

`sshkeys_core` module bumped to 2.2.0

Puppet 7 bumps the `sshkeys_core` modules to 2.2.0 in the Puppet agent. [PA-3473](#)

Call simple server status endpoint

Puppet updates the endpoint for checking the server status to `/status/v1/simple/server`. If the call returns a 404, it makes a new call to `/status/v1/simple/master`, and ensures backwards compatibility. [PUP-10673](#)

Default value of `disable_i18n` changed from false to true

The default value for the `disable_i18n` setting has changed from false to true and locales are not plugin synced when i18n is disabled. [PUP-10610](#)

Pathspec no longer vendored

The `pathspec` Ruby library is no longer vendored in Puppet. If you require this functionality, you need to install the `pathspec` Ruby gem. [PUP-10107](#)

Deprecations and removals

`func3x_check` setting removed

The `func3x_check` setting has been removed. [PUP-10724](#)

`master_used` report parameter removed

The deprecated `master_used` parameter has been removed. Instead use `server_used`. [PUP-10714](#)

`facterng` feature flag removed

The `facterng` feature flag has been removed. It is not needed anymore as Puppet 7 uses Factor 4 by default. [PUP-10605](#)

`held` removed from apt provider

The apt provider no longer accepts deprecated `ensure=held`. Use the `mark` attribute instead. [PUP-10597](#)

Method from `DirectoryService` removed

The deprecated `DirectoryService#write_to_file` method has been removed. [PUP-10489](#)

Method from `Puppet::Provider::NameService` removed

The deprecated `Puppet::Provider::NameService#listbyname` method has been removed. [PUP-10488](#)

Methods from `TypeCalculator` removed

The deprecated `TypeCalculator.enumerable` has been removed, and the functionality has been moved to `Iterable`. [PUP-10487](#)

`Enumeration` type removed

The deprecated `Enumeration` class has been removed, and its functionality has been moved to `Iterable`. [PUP-10486](#)

`Puppet::Util::Yaml.load_file` removed

The deprecated `Puppet::Util::Yaml.load_file` method has been removed. [PUP-10475](#)

`Puppet::Resource` methods removed

The following deprecated `Puppet::Resource` methods have been removed:

- `Puppet::Resource.set_default_parameters`

- `Puppet::Resource.validate_complete`
- `Puppet::Resource::Type.assign_parameter_values`. [PUP-10474](#)

legacy auth.conf support removed

The legacy `auth.conf` has been deprecated for several major releases. Puppet 7 removes all support for legacy `auth.conf`. Instead, authorization to Puppet REST APIs is controlled by `puppetserver auth.conf`. In addition, the `allow` and `deny` rules in `fileserver.conf` are now ignored and Puppet logs an error for each entry. The `rest_authconfig` setting has also been removed. [PUP-10473](#)

Puppet.define_settings removed

The deprecated `Puppet.define_settings` method has been removed. [PUP-10472](#)

Application orchestration language features removed

The deprecated application orchestration language features have been removed. The keywords `application`, `site`, `consumes` and `produces`, and the `export` and `consume` metaparameters, now raise errors. The keywords are still reserved, but can't be used as a custom resource type or attribute name. The environment catalog REST API has also been removed, along with supporting classes, such as the environment compiler and validators. [PUP-10446](#)

Puppet::Network::HTTP::ConnectionAdapter removed

The `Puppet::Network::HTTP::ConnectionAdapter` has been removed, and contains the following breaking changes:

- The Client networking code has been moved to `Puppet::HTTP`.
- The `Puppet::Network::HttpPool.http_instance` method has been removed.
- The `Puppet.lookup(:http_pool)` has been removed.
- The deprecated `Puppet::Network::HttpPool.http_instance` and `connection` methods have been preserved. [PUP-10439](#)

environment_timeout_mode setting removed

The `environment_timeout_mode` setting has been removed. Puppet no longer supports environment timeouts based on when the environment was created. In Puppet 7, the `environment_timeout` setting is always interpreted as 0 (never cache), unlimited (always cache), or from when the environment was last used. [PUP-10619](#)

Networking code from the parent REST terminus removed

The Networking code from the parent REST terminus has been removed, and is a breaking change for any REST terminus that relies on the parent REST terminus to perform the network request and process the response. The REST termini must implement the `find`, `search`, `save` and `destroy` methods for their indirected model. [PUP-10440](#)

Dependency on http-client gem removed

The dependency on the `http-client` gem has been removed. If you have a Puppet provider that relies on this gem, you must install it. [PUP-10490](#)

HTTP file content terminus removed

The HTTP file content terminus has been removed. It is no longer possible to retrieve HTTP file content using the `indirector`. Instead, use Puppet's builtin HTTP client instead: `response = Puppet.runtime[:http].get(URI("http://example.com/path"))`. [PUP-10442](#)

Puppet::Util::HttpProxy.request_with_redirects removed

The `Puppet::Util::HttpProxy.request_with_redirects` method has been removed, and moves the `Puppet::Util::HttpProxy` class to `Puppet::HTTP::Proxy`. The old constant is backwards compatible. [PUP-10441](#)

Puppet::Rest removed

`Puppet::Rest` removed and `Puppet::Network::HTTP::Compression` have been removed. This change moves `Puppet::Network::Resolver` to `Puppet::HTTP::DNS` and deprecates `Puppet::Network::HttpPool` methods. [PUP-10438](#)

Remove strict_hostname_checking removed

The deprecated `strict_hostname_checking` and `node_name` settings have been removed. The functionality of these settings is possible using explicit constructs within a `site.pp` or fully featured `enc`. [PUP-10436](#)

puppet module build, generate and search actions removed

The `puppet module build`, `generate` and `search` actions have been removed. Use Puppet Development Kit (PDK) instead. [PUP-10387](#)

puppet status application has been removed

The deprecated `puppet status` application has been removed. [PUP-10386](#)

The puppet cert and key commands removed

The non-functioning `puppet cert` and `puppet key` commands have been removed. Instead use `puppet ssl` on the agent node and `puppetserver ca` on the CA server. [PUP-10369](#)

SSL code, termini and settings removed

The following SSL code, termini and settings have been removed:

- `Puppet::SSL::Host`
- `Puppet::SSL::Key`
- `Puppet::SSL::{Certificate,CertificateRequest}.indirection`
- `Puppet::SSL::Validator*`
- `ssl_client_ca_auth`
- `ssl_server_ca_auth` [PUP-10252](#)

The func3x_check setting has been removed

The setting to turn off `func 3x` API validation has been removed. Now all 3x functions are validated. [PUP-9469](#)

The future_features logic has been removed

The unused `future_features` setting has been removed. [PUP-9426](#)

The puppet man application has been removed

The `puppet man` application is no longer needed and has been removed. The agent package now installs man pages so that `man puppet` produces useful results. Puppet's help system (`puppet help`) is also available. [PUP-8446](#)

The execfail method from util/execution has been removed

The following deprecated methods have been removed:

- `Puppet::Provider#execfail`
- `Puppet::Util::Execution.execfail`. [PUP-7584](#)

The win32-process has been removed

The Puppet dependency on the win32-process gem has been removed. You can implement the functionality using FFI. [PUP-7445](#)

The win32-service gem has been removed

The dependency on the win32-service gem has been removed and uses the Daemon class in Puppet instead. [PUP-5758](#)

The win32-security gem has been removed from Puppet

To improve Puppet's handling of Unicode user and group names on Windows, some of the code interacting with the Windows API has been rewritten to ensure wide character (UTF-16LE) API variants are called. As a result, Puppet no longer needs the win32-security gem. Any code based references to the gem have been removed. The gem currently remains for backward compatibility, but is to be removed in a future release. [PUP-5735](#)

The capability to install an agent on Windows 2008 and 2008 R2 has been removed

You can no longer install Puppet 7 agents on Windows versions lower than 2012. [PA-3364](#)

Support for Ruby versions older than 2.5 removed

Support for Ruby versions older than 2.5 has been removed, and Fixnum and Bignum have been replaced with Integer. [PUP-10509](#)

dir monkey-patch removed

This external dependency on the win32/dir gem has been removed and replaces CSIDL constants with environment variables. [PUP-10653](#)

Master removed from docs

Documentation for this release replaces the term master with primary server. This change is part of a company-wide effort to remove harmful terminology from our products. For the immediate future, you'll continue to encounter master within the product, for example in parameters, commands, and preconfigured node groups. Where documentation references these codified product elements, we've left the term as-is. As a result of this update, if you've bookmarked or linked to specific sections of a docs page that include master in the URL, you'll need to update your link.

Resolved issues

Puppet agent installation fails when msgpack is enabled on puppetserver

Previously, the agent failed to deserialize the catalog and fail the run if the msgpack gem was enabled but not installed. Now the agent only supports that format when the msgpack gem is installed in the agents vendored Ruby. [PUP-10772](#)

Puppet feature detection leaves Ruby gems in a bad state

This release fixes a Ruby gem caching issue that prevented the agent from applying a catalog if a gem was managed using the native package manager, such as yum or apt. [PUP-10719](#)

Puppet 6 agents do not honor the `usecacheonfailure` setting when using `server_list`

Previously, when `server_list` was used when there was no server accessible, the Puppet run failed even if `usecacheonfailure` was set to true. Now Puppet only fails if `usecacheonfailure` is set to false. [PUP-10648](#)

Setting `certname` in multiple sections bypasses validation

Previously, Puppet only validated the `certname` setting when specified in the main setting, but not if the value was in a non-global setting like `agent`. As a result, it was possible to set the `certname` setting to a value containing uppercase letters and prevent the agent from obtaining a certificate the next time it ran. Puppet now validates the `certname` setting regardless of which setting the value is specified in. [PUP-9481](#)

Issues caused by backup to the local `filebucket`

By default, Puppet won't backup files it overwrites or deletes to the local `filebucket`, due to issues where it became unbounded. You can re-enable the local `filebucket` by setting `File { backup => 'puppet' }` as a resource default. [PUP-9407](#)

Remove future feature flag for `prefetch_failed_providers` in `transaction.rb`

If a provider `prefetch` method raises a `LoadError` or `StandardError`, the resources associated with the provider are marked as failed, but unrelated resources are applied. Previously this behavior was controlled by the `future_features` flag, and disabled by default. [PUP-9405](#)

Change default value of `hostcsr` setting

The default value of the `hostcsr` setting has been updated to match where Puppet stores the certificate request (CSR) when waiting for the CA to issue a certificate. [PUP-9346](#)

Refactor the SMF provider to implement enableable semantics

Previously, the SMF provider did not properly implement enableable semantics. Now `enable` and `ensure` are independent operations where `enable` handles whether a service starts or stops at boot time, and `ensure` handles whether a service starts or stops in the current running instance. [PUP-9051](#)

The list of reserved type names known to the parser validator is incomplete

A class or defined type in top scope can no longer be named `init`, `object`, `sensitive`, `semver`, `semverrange`, `string`, `timestamp`, `timespan` or `typeset`. You can continue to use these names in other scopes such as `mymodule::object`. [PUP-7843](#)

Export or virtualize class error

Previously, Puppet returned a warning or error if it encountered a virtual class or an exported class, but it still included resources from the virtual class in the catalog. Now Puppet always error on virtual and exported classes. [PUP-7582](#)

`Puppet::Util::Windows::String.wide_string` embeds a NULL char

This release removes a Ruby workaround for wide character strings on Windows. [PUP-3970](#)

`puppet config set certname` accepts upper-case names

Previously, the `puppet config set` command could set a value that was invalid, causing Puppet to fail the next time it ran or the service was restarted. Now the command validates the value before committing the change to `puppet.conf`. [PUP-2173](#)

Unable to read `last_run_summary.yaml` from user

Puppet agent code now aligns with the new `last_run_summary.yaml` location. [PA-3253](#)

Puppet known issues

These are the known issues in this version of Puppet.

User and group management on macOS 10.14 and above requires Full Disk Access (FDA)

To manage users and groups with Puppet on macOS 10.14 and above, you must grant Puppet Full Disk Access (FDA). You must also grant FDA to the parent process that triggers your Puppet run. For example:

- To run Puppet in a server-agent infrastructure, you must grant FDA to the `pxp-agent`.
- To run Puppet from a remote machine with SSH commands, you must grant FDA to `sshd`.
- To run Puppet commands from the terminal, you must grant FDA to `terminal.app`.

To give Puppet access, go to **System Preferences > Security & Privacy > Privacy > Full Disk Access**, and add the path to the Puppet executable, along with any other parent processes you use to run. For detailed steps, see [Add full disk access for on 10.14 and newer](#). Alternatively, set up automatic access using Privacy Preferences Control Profiles and a Mobile Device Management Server. [PA-2226](#), [PA-2227](#)

The `puppet node clean` command fails for users who have their `cadir` in the legacy location

In Puppet 7, the default location of the `cadir` has moved. If you have it in the old location, most upgrades trigger a warning when executing commands from Puppet. It causes the `puppet node clean` command to fail. [PUP-10786](#)

Hiera `knockout_prefix` is ineffective in hierarchies more than three levels deep

When specifying a deep merge behaviour in Hiera, the `knockout_prefix` identifier is effective only against values in an adjacent array, and not in hierarchies more than three levels deep. [HI-223](#)

Specify the epoch when using version ranges with the `yum` package provider

When using version ranges with the `yum` package provider, there is a limitation which requires you to specify the epoch as part of the version in the range, otherwise it uses the implicit epoch ``0``. For more information, see the [RPM packaging guide](#). [PUP-10298](#)

Deferred functions can only use built-in Puppet types

Deferred functions can only use types that are built into Puppet (for example `String`). They cannot use types from modules like `stdlib` because Puppet does not plugin-sync these types to the agent. [PUP-8600](#)

The Puppet agent installer fails when `systemd` is not present on Debian 9

The `puppet-agent` package does not include `sysv` init scripts for Debian 9 (Stretch) and newer. If you have disabled or removed `systemd`, `puppet-agent` installation and Puppet agent runs can fail.

Upgrading Windows agent fails with `scriptHalted` error

Registry references to `nssm.exe` were removed in [PA-3263](#). Upgrading from a version without this update to a version that contains it triggers a Windows `SecureRepair` sequence that fails if any of the files delivered in the original `*.msi` package are missing. This is an issue when upgrading to one of the following Puppet agent versions: 5.5.21, 5.5.22, 6.17.0, 6.18.0, 6.19.0, 6.19.1, 6.20.0, 7.0.0, 7.1.0 or 7.3.0. To work around this issue, put the `*.msi` file for the currently installed version in the `C:\Windows\Installer` folder before you upgrade. Starting with Puppet agent 6.21.0 and 7.4.0, the `nssm.exe` registry value will be replaced with an empty string, instead of the registry key being removed, to avoid triggering Windows `SecureRepair`. [PA-3545](#)

The Puppet agent installer fails when `systemd` is not present on Debian 9

In versions less than 7.4.0, the `puppet-agent` package does not include `sysv` init scripts for Debian 9 (Stretch) and newer. If you had disabled or removed `systemd`, the `puppet-agent` installation and agent runs could fail. This is now fixed. [PA-2028](#)

Puppet Server release notes

Puppet Server 7.0.3

Released 9 February 2021

This release updates dependencies to include security fixes.

Puppet Server 7.0.2

Released 20 January 2021

Bug fix

- The warning issued when the CA dir is inside the SSL dir now only prints server logs at startup and when using the `puppetserver ca` CLI, instead of any time a Puppet command is used. ([SERVER-2934](#))

Puppet Server 7.0.1

Released 15 December 2020

Enhancements

- The JRuby version has been bumped from 9.2.13.0 to 9.2.14.0. ([SERVER-2925](#))

Bug fixes

- The CA command line tool now correctly honors the `server` sections in the `puppet.conf`.
- When creating the symlink between the new and legacy `cadirs` the symlink will now be properly owned by the `puppet` user. ([SERVER-2917](#))

Puppet Server 7.0.0

Released 17 November 2020

Puppet Server 7.0 is a major release. It breaks compatibility with agents prior to 4.0 and the legacy Puppet `auth.conf`, moves the default location for the `cadir`, and changes defaults for fact caching and cipher suites. See below for more details. Caution is advised when upgrading.

New features

- The default value for the `cadir` setting is now located at `/etc/puppetlabs/puppetserver/ca`. Previously, the default location was inside Puppet's own `ssldir` at `/etc/puppetlabs/puppet/ssl/ca`. This change makes it safer to delete Puppet's `ssldir` without accidentally deleting your CA certificates.
- The `puppetserver` CA CLI now provides a `migrate` command to move the CA directory from the Puppet `confdir` to the `puppetserver confdir`. It leaves behind a symlink on the old CA location, pointing to the new location at `/etc/puppetlabs/puppetserver/ca`. The symlink provides backwards compatibility for tools still expecting the `cadir` to exist in the old location. In a future release, the `cadir` setting will be removed entirely. ([SERVER-2896](#))
- The default value for the facts cache is now JSON instead of YAML. You can re-enable the old YAML terminus in `routes.yaml`. ([PUP-10656](#))
- Support for legacy Puppet `auth.conf` has been removed and the `jrubby-puppet.use-legacy-auth-conf` setting no longer works. Use Puppet Server's `auth.conf` file instead. ([SERVER-2778](#))
- Puppet Server no longer services requests for legacy (3.x) Puppet endpoints. Puppet Agents before 4.0 are no longer be able to check in. ([SERVER-2791](#))
- This release removes default support for many cipher suites when contacting Puppet Server. The new default supported cipher suites are: `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`,

TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, and TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384. This change aligns open source Puppet with Puppet Enterprise. Note that this change may break on old platforms. To re-enable older cipher suites you may edit the `webserver.conf`. Valid cipher suite names are listed in the [JDK Documentation](#). (SERVER-2913)

- Puppet Server now provides an HTTP client whose API conforms to the HTTP client provided by Puppet. This new client is stored in the Puppet runtime as `Puppet.runtime[:http]`. (SERVER-2780)

Puppet Server known Issues

For a list of all known issues, visit our [Issue Tracker](#).

Cipher updates in Puppet Server 6.5

Puppet Server 6.5 includes an upgrade to the latest release of Jetty's 9.4 series. With this update, you may see "weak cipher" warnings about ciphers that were previously enabled by default. Puppet Server now defaults to stronger FIPS-compliant ciphers, but you must first remove the weak ciphers.

The ciphers previously enabled by default have not been changed, but are considered weak by the updated standards. Remove the weak ciphers by removing the `cipher-suite` configuration section from the `webserver.conf`. After you remove the `cipher-suite`, Puppet Server uses the FIPS-compliant ciphers instead. This release includes the weak ciphers for backward compatibility only.

The FIPS-compliant cipher suites, which are not considered weak, will be the default in a future version of Puppet. To maintain backwards compatibility, Puppet Server explicitly enables all cipher suites that were available as of Puppet Server 6.0. When you upgrade to Puppet Server 6.5.0, this affects you in two ways:

1. The 6.5 package updates the `webserver.conf` file in Puppet Server's `conf.d` directory.
2. When Puppet Server starts or reloads, Jetty warns about weak cipher suites being enabled.

This update also removes the `so-linger-seconds` configuration setting. This setting is now ignored and a warning is issued if it is set. See Jetty's [so-linger-seconds](#) for removal details.

Note: On some older operating systems, you might see additional warnings that newer cipher suites are unavailable. In this case, manage the contents of the `webserver.cipher-suites` configuration value to be those strong suites that available to you.

Server-side Ruby gems might need to be updated for upgrading from JRuby 1.7

When upgrading from Puppet Server 5 using JRuby 1.7 (9k was optional in those releases), Server-side gems that were installed manually with the `puppetserver gem` command or using the `puppetserver_gem` package provider might need to be updated to work with the newer JRuby. In most cases gems do not have APIs that break when upgrading from the Ruby versions implemented between JRuby 1.7 and JRuby 9k, so there might be no necessary updates. However, two notable exceptions are that the `autosign` gem should be 0.1.3 or later and `yard-doc` must be 0.9 or later.

Potential JAVA ARGS settings

If you're working outside of lab environment, increase `ReservedCodeCache` to 512m under normal load. If you're working with 6-12 JRuby instances (or a `max-requests-per-instance` value significantly less than 100k), run with a `ReservedCodeCache` of 1G. Twelve or more JRuby instances in a single server might require 2G or more.

Similar caveats regarding scaling `ReservedCodeCache` might apply if users are managing `MaxMetaspace`.

`tmp` directory mounted `noexec`

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

```
Nov 12 17:46:12 fqdn.com java[56495]: Failed to load feature test for posix:
can't find user for 0
```

```
Nov 12 17:46:12 fqdn.com java[56495]: Cannot run on Microsoft Windows
without the win32-process, win32-dir and win32-service gems: Win32API only
supported on win32
Nov 12 17:46:12 fqdn.com java[56495]: Puppet::Error: Cannot determine basic
system flavour
```

This is caused by the fact that JRuby contains some embedded files which need to be copied somewhere on the filesystem before they can be executed ([see this JRuby issue](#)). To work around this issue, you can either mount the `/tmp` directory without `noexec`, or you can choose a different directory to use as the temporary directory for the Puppet Server process.

Either way, you'll need to set the permissions of the directory to `1777`. This allows the Puppet Server JRuby process to write a file to `/tmp` and then execute it. If permissions are set incorrectly, you'll get a massive stack trace without much useful information in it.

To use a different temporary directory, you can set the following JVM property:

```
-Djava.io.tmpdir=/some/other/temporary/directory
```

When Puppet Server is installed from packages, add this property to the `JAVA_ARGS` and `JAVA_ARGS_CLI` variables defined in either `/etc/sysconfig/puppetserver` or `/etc/default/puppetserver`, depending on your distribution. Invocations of the `gem`, `ruby`, and `irb` subcommands use the updated `JAVA_ARGS_CLI` on their next invocation. The service will need to be restarted in order to re-read the `JAVA_ARGS` variable.

Puppet Server Fails to Connect to Load-Balanced Servers with Different SSL Certificates

SERVER-207: Intermittent SSL connection failures have been seen when Puppet Server tries to make SSL requests to servers via the same virtual ip address. This has been seen when the servers present different certificates during the SSL handshake.

Facter release notes

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

Facter 4.0.51

Released 16 February 2021 and shipped with Puppet Platform 7.4.1.

This release includes minor maintenance changes. For the latest features, see the release notes for Facter 3.14.50.

Facter 4.0.50

Released 9 February 2021 and shipped with Puppet Platform 7.4.0.

Enhancements

- **Networking fact improvements for AIX.** The AIX networking resolver now uses FFI to detect networking interfaces and IPs. Previously, VLANs and secondary IPs were not displayed. [FACT-2878](#)
- **Improved performance for blocking legacy facts.** This release improves the performance of blocking legacy facts by implementing a different mechanism. The new mechanism does not allow legacy groups to be overridden by a group with the same name in `fact-groups`. If you add a legacy group in `fact-groups`, it is ignored. The new implementation is faster for use cases involving multiple custom facts that depend on core facts. [FACT-2917](#)

Resolved issues

- **Facter::Core::Execution does not set status variables in Facter 4.** This release reimplements `Open3.popen3` to use `Process.wait` instead of `Process.detach`. [FACT-2934](#)
- **Facter error message — no implicit conversion of nil into String — when determining processor speed on Linux.** Facter now handles processor speed values where `log10` is not set (3, 6, 9, 12). [FACT-2927](#)

- **Facter fails "closed" if the facter.conf file is invalid.** Previously, Facter failed when an invalid config file was provided. Facter now logs a warning message stating that the parsing of the config file failed and continues retrieving facts with the default options. [FACT-2924](#)
- **Domain on Windows does not prioritise registry.** Facter now prioritises information from registry on Windows, instead of network interface domain names. [FACT-2923](#)
- **LinuxMint Tessa not recognized.** Previously, the `os.release` fact was retrieved from the `/etc/os-release` file, but Facter 3 read other release files based on operating system (OS). Now Facter retrieves `os.release` from the specific release file for every OS. [FACT-2921](#)

Facter 4.0.49

Released 20 January 2021 and shipped with Puppet Platform 7.3.0.

Resolved issues

- **Aggregate facts are broken.** Previously, Facter broke when trying to add a debug message for the location where aggregate facts are resolved from. This only happened with aggregate facts that returned an array or hash without having an aggregate block call. [FACT-2919](#)

Facter 4.0.48

Released 20 January 2021.

Enhancements

- **Rewritten tests for Linux networking resolver.** This release refactors the Linux networking resolver, including fixing the unit tests and adding new ones. [FACT-2901](#)

Resolved issues

- **Facter 4.0.x does not return the domain correctly when set in the registry.** Previously, Facter did not retrieve the domain correctly on Linux and resulted in a faulty FQDN facts. Facter also failed to retrieve domain facts when Windows did not expose the host's primary DNS suffix. This is now fixed. [FACT-2882](#)
- **Legacy group blocks processors core fact.** Blocking legacy facts no longer blocks processors core fact. [FACT-2911](#)
- **Facter Hocon output format.** The `--hocon` option now functions as intended. [FACT-2909](#)
- **Legacy blockdevice vendor and size facts not resolving.** This release fixes `blockdevice_*_size` facts not resolving on AIX, and `blockdevice_*_vendor` facts not resolving on Linux and Solaris. [FACT-2903](#)
- **Facter detects OS family without checking or translating the information.** Previously, Facter detected the OS family by reading `/etc/os-release` without checking or translating the information. This is now fixed and Facter translates the `id_like` field from `/etc/os-release` to Facter known families. [FACT-2902](#)

Facter 4.0.47

Released 15 December 2020 and shipped with Puppet Platform 7.1.0.

New features

- **Added `scope6` fact for per binding.** This release adds the `scope6` fact under every `ipv6` address from the `interface.bindings6` fact. [FACT-2843](#)
- **Support for AWS IMDSv2.** This release updates the EC2 fact to use IMDSv2 to authenticate. To use v2, set the `AWS_IMDSv2` environment variable to `true`. Note that the token is cached for a maximum of 100 seconds.

Resolved issues

- **Facter 4 does not resolve hostname facts.** Previously, Facter failed when `Socket.getaddrinfo` was called, which prevented retrieval of FQDN information. This is now fixed. [FACT-2894](#)
- **Gem-based Facter 4 does not log the facts in debug mode.** This release adds log messages for resolved fact values. [FACT-2883](#)

- **Gem-based Factor 4 does not return the complete FQDN.** Previously, domain was not retrieved correctly on Linux based systems and resulted in a faulty FQDN fact. This release uses Ruby Socket methods to retrieve domain correctly. [FACT-2882](#)
- **Puppet 7 treats non existent facts differently to Puppet 6.** This release excludes custom facts with nil value from `to_user_output`, `values` and `to_hash` Ruby Factor API's. The custom facts with nil value are still returned by `value`, `fact` and `[]` API's. [FACT-2881](#)
- **Facts failing on machines with VLANs.** With this release, `dots` in legacy fact names are ignored. They are not used as an indicator of a fact hierarchy because legacy facts cannot compose and have a flat (key - value) structure. [FACT-2870](#)
- **Factor 4 changes `is_virtual` fact from boolean to string.** This release fixes a regression that caused the `is_virtual` to be a string instead of a boolean. [FACT-2869](#)
- **External facts are loaded when using puppet lookup for a different node.** The `load_external` API method was missing in Factor 4. This is now fixed. [FACT-2859](#)
- **Factor fails when the interface name is not UTF-8.** Previously, a pointer used to indicate networking information was being released by the GC too early and the memory was overridden, resulting in inconsistent data. The fix extends the scope of the pointer so that the memory it points to does not release prematurely. [FACT-2856](#)
- **Failure when a structured custom fact has the wrong layout.** This release adds a log message when custom fact names are incompatible and a fact hierarchy cannot be created. [FACT-2851](#)
- **Cannot retrieve local facts error.** This release fixes an error thrown on systems with a low file descriptor limit. [FACT-2898](#)
- **Dig method fails on Puppet \$facts.** The Factor 4 API method `to_hash` returned a different data type to Factor 3. This release ensures the `to_hash` method returns a Ruby Hash instance. [FACT-2897](#)
- **Factor fails when trying to retrieve ssh facts.** Factor now skips reading ssh keys it does not recognise. [FACT-2896](#)
- **Missing primary interface check on all platforms.** Similar to Factor 3, this release adds a final check that detects the primary interface from the IP. Localhost IPs are excluded. [FACT-2892](#)
- **Factor 4.0.46 breaks virtual flag.** Factor now checks whether the `/proc/lve/list` file is a regular file, instead of checking if it is executable. [FACT-2891](#)
- **Factor 4.0.46 does not load external fact files in lexicographical order.** This is now fixed. [FACT-2874](#)
- **Secondary interfaces are not reported.** `Networking.interfaces` now display secondary interfaces (with or without the label) and the `VLANs.MAC` address is correctly displayed for bonded interfaces. If DHCP is not found, use the `dhcpcd -U <interface_name>` command to search. [FACT-2872](#)

Deprecations and removals

Update rake task for generating facts and tests. The scripts used to generate facts were outdated and had never been used. This release removes them. [FACT-2298](#)

Factor 4.0.46

Released 19 November 2020 and shipped with Puppet Platform 7.4.1.

New features

- **Operating system hierarchy.** Factor 4 introduces a hierarchy for operating systems. The hierarchy allows you to load facts from the child and all parents. If the same fact is present in a child and a parent, the one from the child takes precedence. [FACT-2555](#)
- **Rake task for mapping fact name and fact class.** To improve the visibility of which facts are loaded for an operating system, Factor 4 has a rake task that prints all facts and the class that resolved that fact. [FACT-2557](#)
- **Blocklist.** Factor 4 allows you to block facts at a granular level — you can block any fact from the fact hierarchy, for both groups of facts and individual facts. [FACT-1976](#)
- **Block legacy facts.** Legacy facts are a subtype of core facts and you can now block them, like any core fact, using the `blocklist` from `factor.conf`. [FACT-2259](#)

- **Block custom facts.** Facter 4 allows you to block custom facts. You can add the fact to the `blocklist` from `facter.conf`. [FACT-2718](#)
- **Block external facts.** Facter 4 allows you to block external facts. Blocking external facts is different from blocking core and custom facts — you need to specify the name of the file from which external facts are loaded to `blocklist` in `facter.conf`. [FACT-2717](#)
- **The puppet facts show command.** The `facter -p` and `facter --puppet` commands have been replaced with `puppet facts show`. [FACT-2719](#)
- **Group for legacy facts.** The legacy group contains all the legacy facts. You can find it in `lib/facter/config.rb` and can block it in `facter.conf`. [FACT-2296](#)
- **The fact-groups group.** You can define your own custom groups in `facter.conf` using the new `fact-groups` group. The `blocklist` and `ttls` groups from `facter.conf` accept predefined groups, custom groups or fact names. You must use the file name when using external facts. [FACT-2331](#)
- **Define fact groups for blocking or caching.** You can define new fact groups in `facter.conf`, and use the group to block or cache facts. [FACT-2515](#)
- **External fact caching.** To cache external facts, use the filename of the external fact when setting the `ttls` in `facter.conf`. [FACT-2619](#)

Enhancements

- **CLI compatible with Facter 3.** Facter 4 reimplemented the Facter 3 command line interface using `thor` gem. [FACT-1955](#)
- **Loaded facts based on operating system hierarchy.** To improve performance, Facter 4 only loads the files and facts that are needed for the operating system it is running on. [FACT-2093](#)
- **Log class name in log messages.** Improved logging in Facter 4 prints the class from which the log was generated. [FACT-2036](#)
- **Restored --timing option to native facter.** The restored `--timing` and `-t` arguments allow you to see how much time it took each fact to resolve. [FACT-1380](#)
- **(Experimental). Reverted parallel resolution of facts.** To improve performance, facts are resolved in parallel on JRuby. [FACT-2819](#)
- **Timeout on resolution.** You can now specify the `timeout` attribute in custom fact options. [FACT-2643](#)
- **Caching core facts .** To cache core facts, add the fact group to `facter.conf` `ttls`. Fact values are stored and retrieved on future runs. After the `ttls` expires, the fact is refreshed. [FACT-2486](#)

Resolved issues

- **Fix Ruby 2.7 warning on Facter 4.** Facter 4 now supports Ruby 2.7. [FACT-2649](#)
- **Facter does not support timeout for shell calls.** External commands have a timeout, and if they do not complete in the given time, they are forced stop. The default timeout is 300 seconds. You can now specify a timeout using the `limit` attribute in `Facter::Core::Execution.execute`. [FACT-2793](#)
- **Fact names are treated as regex and can lead to caching of unwanted facts.** The `regex` used to detect facts has been improved to distinguish between fact groups and legacy facts. [FACT-2787](#)
- **Facter uptime shows host uptime inside docker container.** Previously, the kernel only reported the host uptime inside a Docker container. You can now see the container uptime. [FACT-2737](#)
- **A fact present in two groups does not get cached if the second groups has a ttls.** If a fact is present in two groups, and both of them have a `ttls` defined in `facter.conf`, the lowest `ttls` is takes precedence. [FACT-2786](#)

Factor known issues

These are the known issues in this version of Factor.

The implementation of the dot notation is not compatible with the Puppet ecosystem

The implementation of the [dot notation feature](#) in Factor 4 is not compatible with other parts of the Puppet ecosystem, and does not give the fact the same name as Factor 3. Factor 4.1 will revert back to the way Factor 3 handled facts with dotted names. [FACT-3004](#)

Incorrect OS description output on Debian 9

Running the `factor` command in Factor 4 returns incorrect output for the `os.distro.description` fact on Debian 9. These facts work correctly in Factor 3 and appear differently when running the `puppet facts diff` command in Factor 4. [FACT-2963](#)

What's new since Puppet 6?

These are the major new features, enhancements, deprecations, and removals since the Puppet 6 release.

Note that this list is not intended to be exhaustive. For all changes, see the [release notes](#).

Get familiar with the latest hardware requirements, supported operating systems and browsers, and network configuration details in [System requirements](#).

Factor 4

Factor 4 introduces new features, including granular blocking and caching of all types of facts, user defined fact groups, fact hierarchies using the `dot` notation and profiling using the `--timing` option. Factor 4 is written in Ruby, instead of C++. It is API compatible with Factor 3, but there may be some inconsistencies. Puppet 7 drops support for Factor 3. For a full list of new features and enhancements, see the [Factor 4 release notes](#).

Ruby 2.7

Puppet 7 agents have upgraded to Ruby 2.7 and dropped support for Ruby 2.3 and 2.4. After upgrading to Puppet 7, use the `puppet_gem` provider to ensure all your gems are installed.

Postgres 11+

PuppetDB now requires Postgres 11+, which allows us to write faster migrations and take advantage of newer features like logical partitioning.

Environment caching

The Puppet 7 `environment_timeout` behaves differently for values that are not 0 or unlimited. Puppet Server keeps your most actively used environments cached, but allows testing environments to fall out of the cache and reduce memory usage

SHA256

Puppet 7 defaults to using SHA256 for all digest operations. MD5 is still be available for non-FIPS platforms, but you must opt into it using the `checksum` parameter for a `file` resource.



CAUTION: To avoid breaking changes when upgrading, either disable remote filebuckets or make sure the agent has the same digest algorithm as server by changing the `digest_algorithm` setting on the agent to `sha256`.

CA directory

We have made changes to prevent you from accidentally deleting your CA directory. This change is backwards compatible.

Puppet language enhancements

The Puppet 7 compiler raises syntax errors if it encounters application orchestration language keywords. These keywords remain reserved for future use.

Platform end-of-life

We have dropped agent support for the following platforms: EL5, Debian 8, SLES 11, Ubuntu 14.04, and Windows 2008/2008R2. Puppet Server and PuppetDB platforms have not changed.

Removals

We've removed the following deprecated functionality:

- Legacy authorization — replaced by Puppet Server's `auth.conf` in Puppet 5.
- Legacy routes — Puppet 3 agents are no longer be able to communicate with these.
- Puppet `key`, `cert` and `status` commands.

For a full list of removals and deprecations, see the [Puppet 7 release notes](#).

Documentation terminology changes

Documentation for this release replaces the term `master` with `primary server`. This change is part of a company-wide effort to remove harmful terminology from our products. For the immediate future, you'll continue to encounter `master` within the product, for example in parameters, commands, and preconfigured node groups. Where documentation references these codified product elements, we've left the term as-is. As a result of this update, if you've bookmarked or linked to specific sections of a docs page that include `master` in the URL, you'll need to update your link.

Getting started with Puppet Enterprise

Puppet Enterprise (PE) is automation software that helps you and your organization be productive and agile while managing your IT infrastructure.

PE is a commercial version of Puppet, our original open source product used by individuals managing smaller infrastructures. It has all the power and control of Puppet, plus a graphical user interface, orchestration services, role-based access control, reporting, and the capacity to manage thousands of nodes. PE incorporates other Puppet-related tools and products to deliver comprehensive configuration management capabilities.

There are two things you need to get started with PE: your content and the Puppet platform.

Content

You develop and store your automation content in a Git repository and upload it onto the Puppet platform. It consists of Puppet code, plan and task code, and Hiera data. You store content in a *control repo*, which contains bundles of code called *modules* and references to additional content from external sources, like the [Puppet Forge](#) — a repository of thousands of modules made by Puppet developers and the Puppet community.

Puppet platform

The Puppet platform includes the primary server, compilers, and agents. Use it to assign your desired state to managed systems, orchestrate ad-hoc automation tasks on managed and unmanaged systems, and get reports about configuration automation activity.

In this guide, you will learn how to install PE, add nodes to the console, set up your control repo, and run through an example task of managing webserver configurations — using either Apache to manage a *nix machine or IIS to manage a Windows machine.

Before you begin, check out our video for more information about how Puppet works.

- [Install PE](#) on page 58

Installing PE sets up a standard installation, which you can use to try out PE with up to 10 nodes or to manage up to 4,000 nodes.

- [Add nodes to the inventory](#) on page 59

Your inventory is the list of nodes managed by Puppet. Add nodes with agents, agentless nodes that connect over SSH or WinRM, or add network devices like network switches and firewalls. Agent nodes help keep your infrastructure in your desired state. Agentless nodes do not have an agent installed, but can do things like run tasks and plans.

- [Add code and set up Code Manager](#) on page 61

Set up your control repo, create a Puppetfile, and configure Code Manager so you can start adding content to your PE environments.

- [Manage Apache configuration on *nix targets](#) on page 65
- [Manage IIS configuration on Windows targets](#) on page 70
- [Next steps](#) on page 76

Now that you have set up some basic automated configuration management with PE, here are some things to do next:

Install PE

Installing PE sets up a standard installation, which you can use to try out PE with up to 10 nodes or to manage up to 4,000 nodes.

Ensure your system capacity can manage a PE installation by reviewing the [hardware requirements for standard installations](#).

A standard PE installation consists of these components installed on a single node:

- The primary server: the central hub of activity, where Puppet code is compiled to create agent catalogs, and where SSL certificates are verified and signed.
- The console: the graphical web interface, which features configuration and reporting tools.
- PuppetDB: the data store for data generated throughout your Puppet infrastructure.

Important: The primary server runs only on *nix machines. Windows machines can run as Puppet agents, and you can manage them with your *nix primary server. If you want to operate a *nix primary server remotely from a Windows machine, before installing PE, configure an SSH client, such as [PuTTY](#), with the hostname or IP address and port of the *nix machine that you want to install as your primary server. When you open an SSH session to install the *nix primary server, log in as the `root` user.

Install PE

Installation uses default settings to install all of the PE infrastructure components on a single node. After installing, you can scale or customize your installation as needed.

Important: Perform these steps on your target primary server logged in as root. If you're installing on a system that doesn't enable root login, switch to the root user with this command: `sudo su -`

1. [Download](#) the tarball appropriate to your operating system and architecture.

Tip: To download packages from the command line, run `wget --content-disposition <URL>` or `curl -JLO <URL>`, using the URL for the tarball you want to download.

2. Unpack the installation tarball:

```
tar -xf <TARBALL_FILENAME>
```

3. From the installer directory, run the installer and follow the CLI instructions to complete your installation:

```
./puppet-enterprise-installer
```

4. Optional: Restart the shell in order to use client tool commands.

Log into the PE console

The console is a graphical interface where you can manage your infrastructure without relying on the command line.

To log in for the first time:

1. Open the console by entering the URL `<PRIMARY_HOSTNAME>` into your browser, where hostname is the trusted certificate name of your primary server.

Note: You'll receive a browser warning about an untrusted certificate because you were the signing authority for the console's certificate, and your Puppet Enterprise deployment is not known to your browser as a valid signing authority. Ignore the warning and accept the certificate.
2. On the login page for the console, log in with the username `admin` and the password you created when installing. Keep track of this login as you will use it later.

Next, check the status of your primary server.

Check the status of your primary server

You can run a task to check the status of your primary server in the console.

A *task* is a single action that allows you to do ad-hoc things like upgrade packages and restart services on target machines. PE comes with a few tasks installed, such as `package`, `service`, and `puppet_conf`, and you can download more tasks from the Forge or write your own.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Task** field, select `service` because you are checking the status of the primary server service.
4. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
5. Under **Task parameters**, parameters and values for the task. The `service` task has two required parameters. For **action**, choose `status`. For **name**, enter `puppet`.
6. Under **Select targets**, select **Node list**.
 - a) In the **Inventory nodes** field, add the hostname of your primary server and select it.
7. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

View the task status and output in the **Jobs** page when it is finished running.

The status of your primary server should be “running” and “enabled”. To learn more about tasks, including how to install them from the Forge or how to write your own, visit the [Installing tasks](#) and [Writing tasks](#) sections of the docs.

Next, use the console to add nodes to your inventory.

Add nodes to the inventory

Your inventory is the list of nodes managed by Puppet. Add nodes with agents, agentless nodes that connect over SSH or WinRM, or add network devices like network switches and firewalls. Agent nodes help keep your infrastructure in your desired state. Agentless nodes do not have an agent installed, but can do things like run tasks and plans.

Add agent nodes

Use the console to add agent nodes to your inventory. Agents help with configuration management by periodically correcting changes to resources and reporting information to the primary server about your infrastructure. Note that adding an agent node installs the agent on the node.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Install agents**.
3. Select the transport method. This connection is used to remotely install the agent on the target node.
 - **SSH** for *nix targets
 - **WinRM** for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click **Add nodes**.

Tip: Click **Installation job started** to view the job details for the task.

Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

Add agentless nodes

Add nodes that will not or cannot have an agent installed on them. Agentless automation allows you to do things like update a package or restart a server on demand for node targets that don't have software installed.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect over SSH or WinRM**.
3. Select a transport method.
 - **SSH** for *nix targets
 - **WinRM** for Windows targets
4. Enter target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Optional: Select additional target options. For example, to add a target port number, select **Target Port** from the drop-down list, enter the number, and click **Add**. For details, see [Transport configuration options](#) on page 313.
6. Click **Add nodes**.

After the nodes have been added to the inventory, they are added to PuppetDB, and you can view them from the **Nodes** page. Nodes in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory node's connection options, or to remove the node from inventory, go to the **Connections** tab on the node's details page.

Managing certificate signing requests in the console

A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

- To manage requests individually, click **Accept** or **Reject**.
- To manage the entire list of requests, click **Accept All** or **Reject All**. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

Related information

[Running Puppet on demand](#) on page 476

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

Add code and set up Code Manager

Set up your control repo, create a Puppetfile, and configure Code Manager so you can start adding content to your PE environments.

The *control repo* is where you store your code. Code in your control repo is usually bundled in modules.

The *Puppetfile* specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Code Manager automates the management and deployment of your Puppet code. It isn't required to use PE, but it is helpful for ensuring Puppet syncs code to your primary server and all your servers run new code at the same time.

Create a control repo from the Puppet template

To create a control repo that includes a standard recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

To base your control repo on the Puppet [control repository template](#), you copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

The control repo template contains the files needed to get you started with a functioning control repo, including:

- An `environment.conf` file to implement a `site-modules/` directory for roles, profiles, and custom modules.
- `config_version` scripts to notify you which control repo version was applied to the agents.
- Basic code examples for setting up roles and profiles.
- An example `hieradata` directory that matches the default hierarchy .
- A Puppetfile to manage content maintained in your environment.

Set up a private SSH key so that your primary server can identify itself to your Git host.

1. Generate a private SSH key to allow access to the control repository.

This SSH key cannot require a password.

- a) Generate the key pair:

```
ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
```

- b) Set appropriate permissions so that the `pe-puppet` user can access the key:

```
puppet infrastructure configure
```

Your keys are now located in `/etc/puppetlabs/puppetserver/ssh`:

- Private key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
- Public key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub`

Configure your Git host to use the SSH public key you generated. The process to do this is different for every Git host. Usually, you create a user or service account, and then assign the SSH public key to it.

Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

See the your Git host docs for detailed instructions on adding SSH keys to your Git server. Commonly used Git hosts include:

- [GitHub](#)
- [BitBucket Server](#)
- [GitLab](#)

2. Create a repository in your Git account, with the name you want your control repo to have.

Steps for creating a repository vary, depending on what Git host you are using (GitHub, GitLab, Bitbucket, or another provider). See your Git host's documentation for complete instructions.

For example, on GitHub:

- a) Click + at the top of the page, and choose **New repository**.
- b) Select the account **Owner** for the repository.
- c) Name the repository (for example, `control_repo`).
- d) Note the repository's SSH URL for later use.

3. If you don't already have Git installed, run the following command on your primary server:

```
yum install git
```

4. From the command line, clone the Puppet `control_repo` template.

```
git clone https://github.com/puppetlabs/control_repo.git
```

5. Change directory into your control repo.

```
cd <NAME OF YOUR CONTROL REPO>
```

6. Remove the template repository as your default source.

```
git remote remove origin
```

7. Add the control repository you created as the default source.

```
git remote add origin <URL OF YOUR GIT REPOSITORY>
```

8. Push the contents of the cloned control repo to your remote copy of the control repo:

```
git push origin production
```

You now have a control repository based on the Puppet `control-repo` template. When you make changes to this repo on your workstation and push those changes to the remote copy of the control repo on your Git server, Code Manager deploys your infrastructure changes.

You also now have a Puppetfile available for you to start adding and managing content, like module code.

Configure Code Manager

Code Manager stages, commits, and synchronizes your code, automatically managing your environments and modules when you make changes.

Enable Code Manager

To enable Code Manager set parameters in the console.

Before you begin

See [Configure settings using the console](#) on page 160 for more details about setting parameters in the console.

1. In the console, click **Node groups**, set the following parameters in the `puppet_enterprise::profile::master` class in the **PE Master** node group.
 - `code_manager_auto_configure` to `true`.
 - `r10k_remote`: This is the location of your control repository. Enter a string that is a valid URL for your Git control repository. For example: `git@<YOUR.GIT.SERVER.COM>:puppet/control.git`.
 - `r10k_private_key`: Enter a string specifying the path to the SSH private key that permits the `pe-puppet` user to access your Git repositories. This file must be located on the primary server, owned by the `pe-puppet` user, and located in a directory that the `pe-puppet` user has permission to view. We recommend `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
2. Click **Commit**.
3. On the command line, run `puppet job run --nodes <NODE NAME>` where `<NODE NAME>` is the name of your primary server.

```
puppet job run --nodes small-doubt.delivery.puppetlabs.net
```

Note: `r10k` is a code management tool that allows you to manage your environment configurations — production, testing, and development — in a source control repository.

Next, test the connection to the control repo.

Set up authentication for Code Manager

To securely deploy environments, Code Manager needs an authentication token for both authentication and authorization.

To generate a token for Code Manager:

1. Assign a user to the deployment role.
2. In the console, create a deployment user.

Tip: Create a dedicated deployment user for Code Manager use.
3. Add the deployment user to the **Code Deployers** role.

Note: This role is automatically created on install, with default permissions for code deployment and token lifetime management.
4. Create a password by clicking **Generate Password**.
5. [Request an authentication token for deployments](#) on page 64

Related information

[Configure puppet-access](#) on page 218

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. If you want to use `puppet-access`, ensure it is configured correctly before using it to generate authentication tokens.

Assign a user to the deployment role

To request an authentication token, you must first assign a user the correct permissions with role-based access control (RBAC).

1. In the console, create a deployment user. We recommend that you create a dedicated deployment user for Code Manager use.
2. Add the deployment user to the **Code Deployers** role. This role is automatically created on install, with default permissions for code deployment and token lifetime management.
3. Create a password by clicking **Generate Password**.

Request the authentication token.

Related information

[Add a user to a user role](#) on page 203

When you add users to a role, the user gains the permissions that are applied to that role. A user can't do anything in PE until they have been assigned to a role.

[Assign a user group to a user role](#) on page 210

After you've imported a group, you can assign it a user role, which gives each group member the permissions associated with that role. You can add user groups to existing roles, or you can create a new role, and then add the group to the new role.

Request an authentication token for deployments

Request an authentication token for the deployment user to enable secure deployment of your code.

By default, authentication tokens have a one-hour lifetime. With the `Override default expiry` permission set, you can change the lifetime of the token to a duration better suited for a long-running, automated process.

Generate the authentication token using the `puppet-access` command.

1. From the command line on the primary server, run `puppet-access login --lifetime 180d`. This command both requests the token and sets the token lifetime to 180 days.

Tip: You can add flags to the request specifying additional settings such as the token file's location or the URL for your RBAC API. See [Configuration file settings for puppet-access](#).

2. Enter the username and password of the deployment user when prompted.

The generated token is stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. To view the token, run `puppet-access show`.

Test the connection to the control repo.

Related information

[Set a token-specific lifetime](#) on page 222

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

[Generate a token for use by a service](#) on page 222

If you need to generate a token for use by a service and the token doesn't need to be saved, use the `--print` option.

Deploy your code

Use the command line to trigger Code Manager after making changes to your Puppetfile.

When you make changes to your Puppetfile, like adding a new module or creating a repo, you must deploy your code before Code Manager can recognize or start managing the content.

SSH into your primary server and run `puppet-code deploy --all --wait`.

You have deployed code to all environments. The `--wait` flag returns results after the deployment is finished. Use the command `puppet-code deploy <ENVIRONMENT>` to deploy code to only a specific environment. You can also deploy code using a [webhook](#) or [custom scripts](#).

Manage Apache configuration on *nix targets

- [Add a module](#) on page 65

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, like installing and configuring a piece of software. You can write your own modules or download pre-built modules from the Forge. While you can use any module available on the Forge, PE customers can take advantage of supported modules. These modules are designed to make common services easier, and are tested and maintained by Puppet. A lot of your infrastructure will be supported by modules, so it is important to learn how to install, build, and use them.

- [Configure your desired state](#) on page 66

Tell PE how your infrastructure should be configured by grouping and classifying nodes based on their function. Before you begin, think of which of your inventory nodes you want to configure with Apache services.

- [Organize webserver configurations with roles and profiles](#) on page 67

The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

Add a module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, like installing and configuring a piece of software. You can write your own modules or download pre-built modules from the Forge. While you can use any module available on the Forge, PE customers can take advantage of supported modules. These modules are designed to make common services easier, and are tested and maintained by Puppet. A lot of your infrastructure will be supported by modules, so it is important to learn how to install, build, and use them.

Install the apache module

To get you started working with modules, install the `puppetlabs/apache` module, which automates installing, configuring, and managing Apache services.

Before you begin

These instructions assume you've installed PE and at least one *nix agent node.

1. Go to the [apache module in the Forge](#).
2. Follow the instructions in the **r10k or Code Manager** drop down to add the module declaration to your Puppetfile.

If you don't specify options, Code Manager installs the latest version and does not update it automatically. To always have the latest version installed, specify `:latest` and it will update automatically when a new version is released. To install a specific version of the module that does not update automatically, specify the version number as a string.

For example, to install version 5.4.0 of the `apache` module that will not update, and its dependencies, use the following code:

```
mod 'puppetlabs/apache', '5.4.0'
mod 'puppetlabs/stdlib', '4.13.1'
mod 'puppetlabs/concat', '2.2.1'
```

The `puppetlabs/stdlib` and `puppetlabs/concat` modules are dependencies of the Apache module. You can see the dependencies for each module by clicking the **Dependencies** tab on a [module's Forge page](#).

3. SSH into your primary server and deploy code by running the `puppet-code deploy --all` command.

You have just installed a Puppet module!

View the apache task in the console

When you successfully install a module, you can start running tasks that are included in the module from the console. The puppetlabs/apache module contains a task that allows you to perform apache service functions.

To see the task in the console:

In the **Run** section, under **Task**, type apache in the **Task** field.

You can now run the apache task.

If you don't see the apache task, try refreshing the console in the browser. If it still does not populate, double check the puppetlabs/apache module is in your Puppetfile and try again.

Configure your desired state

Tell PE how your infrastructure should be configured by grouping and classifying nodes based on their function. Before you begin, think of which of your inventory nodes you want to configure with Apacheservices.

Create your classification group

Create a classification node group. This is a parent group to other node groups you create that contain classification data. You only need to set this up one time.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group:
 - **Parent name:** All Nodes
 - **Group name:** All Classification
 - **Environment:** production
 - **Environment group:** do not select
3. Click **Add**.

You have added a classification group. Next, create a node group and add it as a child of this classification group.

Create your apache node group

After you create your parent classification group, make the apache node group a child of it.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group:
 - **Parent name:** All Classification
 - **Group name:** apache
 - **Environment:** production
 - **Environment group:** do not select
3. Click **Add**.

You now have a node group for nodes you want to configure Apache on. Next, determine which nodes from your inventory you want to run Apache on and add them to the apache node group.

Add nodes to the apache node group

Once you determine which nodes from your inventory should be in your apache node group, pin the nodes to the group. Pinning allows you to add nodes to a group one at a time. You can also dynamically add nodes to a node group based on rules you set in the console, which is helpful for large inventories.

If you are stuck on deciding which nodes should be in your apache node group, read about [Best practices for classifying node groups](#) on page 318.

1. In the console, click **Node groups** and select the apache node group you just created.
2. On the **Rules** tab, under **Certname**, enter the certname of a node you want to add to the apache node group.

3. Click **Pin node**.
4. Repeat for additional nodes you want to add to the `apache` group.
5. When you are done pinning nodes, commit changes.

You have added your `apache` nodes into a node group. For more information on adding and classifying nodes, read about [Making changes to node groups](#) on page 326 and [Grouping and classifying nodes](#) on page 318.

Run Puppet on your Apache nodes

Run Puppet in the console to enforce your desired state on the `apache` node group you created.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Under **Run options**, do not select anything.
3. From the list of target types, select **Node group**.
4. In the **Chose a node group box**, search for the `apache` node group and click **Select**.
5. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only on the nodes that failed during the initial run.

Congratulations! You have run Puppet on your `apache` node group. In the future, the `apache` nodes will check in with PE every 30 minutes to confirm that their `apache` configuration matches what you've specified, and PE will correct it if it doesn't.

Next, learn how to use roles and profiles to efficiently manage system configurations.

Organize webserver configurations with roles and profiles

The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

Roles and profiles help you pick out relevant pieces of code from modules and bundle them together to create your own custom set of code for managing things. *Profiles* are the actual bundles of code. *Roles* gather profiles together so that you can assign them to nodes. This allows you to efficiently organize your Puppet code.

To learn about roles and profiles by example, follow these instructions to define a profile that configures virtual webhost (`vhost`) to serve the `example.com` website and include a firewall rule. Then, you will create a role to contain the profile, and you'll assign it to the `apache` node group that you created earlier. This lays down a base structure where, if you had additional websites to serve, you would create additional profiles for them, and those profiles could be separated or combined inside the roles as needed.

Because you are adding a firewall rule, make sure you add the [puppetlabs/firewall](#) module to your Puppetfile, following the same process you used to add the `apache` module. Remember to add the firewall modules dependencies — `puppetlabs-stdlib`.

```
mod 'puppetlabs/firewall', '2.3.2'
mod 'puppetlabs/stdlib' , '4.0.0'
```

See [The roles and profiles method](#) on page 360 for more information about how roles and profiles work.

Set up your prerequisites

Before you begin writing content for roles and profiles, you need to create modules to store them in.

1. Create one module for `profile` and one for `role` directly in your control repo. Do not put them in your Puppetfile.
2. Make a new directory in the repo named `site`. For example, `/etc/puppetlabs/code/environments/production/site`.
3. Edit the `environment.conf` file to add `site` to the `modulepath`. This is the place where Puppet looks for module information. For example: `modulepath = site:modules:$basemodulepath`.
4. Put the `role` and `profile` modules in the `site` directory.

Write a profile for your Apache vhost

Write a webserver profile that includes rules for your Apache vhost and firewall.

Before you begin

Make sure you have already installed the `puppetlabs/apache` module and the `puppetlabs/firewall` module from the forge.

Note: We recommend writing your code in a code editor, such as VSCode, and then pushing to your Git server. Puppet has an [extension](#) for VSCode that supports syntax highlighting of the Puppet language.

1. In the profile module you added, create the following directory:

- manifests/
 - webserver/
 - example.pp

2. Paste the following Puppet code into the new `example.pp` file:

```
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
webserver/example.pp
class profile::webserver::example (
  String $content      = "Hello from vhost\\n",
  Array[String] $ports = ['80']
  Array[String] $ips    = ['127.0.0.1', '127.0.0.2'],
)
{
  class { 'apache':
    default_vhost => false,
  }
  {
    apache::vhost { 'example.com':
      port      => $ports,
      ip        => $ips,
      ip_based  => true,
      docroot   => '/var/www/html',
    }
    file { '/var/www/html/index.html':
      ensure => file,
      content => $content,
    }
    firewall { '100 allow http and https access':
      dport => $ports,
      proto => tcp,
      action => accept
    }
  }
}
```

This profile applies custom rules for the `apache::vhost` class that include arrays of `$ports` and `$ips`. The code uses `file` to ensure there is content on the main page of your vhost. Finally, there is a firewall rule that only allows traffic from the port or ports set in the `$ports` array.

You can add your own code to the profile as needed. Look at the readme and reference sections for the `puppetlabs/apache` and `puppetlabs/firewall` modules in the Forge for more content.

Set data for the profile

Hiera is a configuration method that allows you to set defaults in your code, or override those defaults in certain circumstances. Use it to fine-tune data within your profile.

Suppose you want to use the custom fact `stage` to represent the deployment stage of the node, which can be `dev`, `test`, or `prod`. For this example, use `dev` and `prod`.

With Hiera structured data, you can set up a four-layer hierarchy that consists of the following:

- `console_data` for data defined in the console.
- `nodes/%{trusted.certname}` for per-node overrides.
- `stage/%{facts.stage}` for setting stage-specific data.
- `common` for global fallback data.

This structure lets you tune the settings for ports and IPs in each stage.

For example, to configure webservers in the development environment to have a custom message and to use port 8080, you'd create a data file with the following name, location, and code content:

```
# cat /etc/puppetlabs/code/environments/production/data/stage/dev.yaml
---
profile::webserver::example::content: "Hello from dev\n"
profile::webserver::example::ports:
- '8080'
```

To have webservers in the production environment listen to all interfaces:

```
# cat /etc/puppetlabs/code/environments/production/data/stage/prod.yaml
---
profile::webserver::example::ips:
- '0.0.0.0'
- ':::'
```

This is the briefest of introductions to all the things you can do with structured data in Hiera. To learn more about setting up hierarchical data, see [Getting started with Hiera](#).

Write a role for your Apache web server

To write roles, think about what machines you'll be managing and decide what else they need in addition to the `webserver` profile.

Say you want all of the nodes in your `apache` node group to use the profile you just wrote. Suppose also that your organization assigns all machines, including workstations, with a profile called `profile::base`, which manages basic policies and uses some conditional logic to include operating-system-specific configuration.

Inside the following file in your control repo:

```
/etc/puppetlabs/code/environments/production/site/role/manifests/
exampleserver.pp
```

Write a role that includes both the base profile and your `webserver` profile:

```
class role::exampleserver {
  include profile::base
  include profile::webserver
}
```

You can add more profiles to this role, or create additional roles with more profile configurations based on your needs.

Assign the role to nodes

Assign the `exampleserver` role to nodes where you want to manage the configuration of the Apache vhost, based on what you wrote in the `profile::webserver::example` profile.

For this example, assume you want to add `role::exampleserver` to all the nodes in the `apache` node group you created.

1. In the console, click **Node groups** and select the `apache` node group.
2. On the **Classes** tab, select `role::exampleserver` and click **Add class**.
3. Commit the change.

Your `apache` node group is managing your Apache vhost based on the rules you coded into your `webserver` profile.

Manage IIS configuration on Windows targets

- [Add a module](#) on page 65

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, like installing and configuring a piece of software. You can write your own modules or download pre-built modules from the Forge. While you can use any module available on the Forge, PE customers can take advantage of supported modules. These modules are designed to make common services easier, and are tested and maintained by Puppet. A lot of your infrastructure will be supported by modules, so it is important to learn how to install, build, and use them.

- [Configure your desired state](#) on page 71

Tell PE how your infrastructure should be configured by grouping and classifying nodes based on their function. Before you begin, think of which of your inventory nodes you want to configure with IIS services.

- [Organize webserver configurations with roles and profiles](#) on page 72

The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

Add a module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, like installing and configuring a piece of software. You can write your own modules or download pre-built modules from the Forge. While you can use any module available on the Forge, PE customers can take advantage of supported modules. These modules are designed to make common services easier, and are tested and maintained by Puppet. A lot of your infrastructure will be supported by modules, so it is important to learn how to install, build, and use them.

Install the IIS module

To get you started working with modules, install the `puppetlabs/iis` module, which automates installing, configuring, and managing IIS services.

Before you begin

These instructions assume you've installed PE and at least one Windows agent node.

1. Go to the [IIS module in the Forge](#).
2. Follow the instructions in the **r10k or Code Manager** drop down to add the module declaration to your Puppetfile.

If you don't specify options, Code Manager installs the latest version and does not update it automatically. To always have the latest version installed, specify `:latest` and it will update automatically when a new version is released. To install a specific version of the module that does not update automatically, specify the version number as a string.

For example, to install version 7.0.0 of the IIS module that will not update, and its dependencies, use the following code:

```
mod 'puppetlabs/iis', '7.0.0'
mod 'puppetlabs/pwshlib', '0.4.0'
```

The `puppetlabs/pwshlib` module is a dependency of the IIS module. You can see the dependencies for each module by clicking the **Dependencies** tab on a [module's Forge page](#).

3. SSH into your primary server and deploy code running the `puppet-code deploy --all` command.

You have just installed a Puppet module!

Configure your desired state

Tell PE how your infrastructure should be configured by grouping and classifying nodes based on their function. Before you begin, think of which of your inventory nodes you want to configure with IIS services.

Create your classification group

Create a classification node group. This is a parent group to other node groups you create that contain classification data. You only need to set this up one time.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group:
 - **Parent name:** All Nodes
 - **Group name:** All Classification
 - **Environment:** production
 - **Environment group:** do not select
3. Click **Add**.

You have added a classification group. Next, create a node group and add it as a child of this classification group.

Create your iis node group

After you create your parent classification group, make the `iis` node group a child of it.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group:
 - **Parent name:** All Classification
 - **Group name:** `iis`
 - **Environment:** production
 - **Environment group:** do not select
3. Click **Add**.

You now have a node group for nodes you want to configure IIS on. Next, determine which nodes from your inventory you want to run IIS on and add them to the `iis` node group.

Add nodes to the iis node group

Once you determine which nodes from your inventory should be in your `iis` node group, pin the nodes to the group. Pinning allows you to add nodes to a group one at a time. You can also dynamically add nodes to a node group based on rules you set in the console, which is helpful for large inventories.

If you are stuck on deciding which nodes should be in your `iis` node group, read about [Best practices for classifying node groups](#) on page 318.

1. In the console, click **Node groups** and select the `iis` node group you just created.
2. On the **Rules** tab, under **Certname**, enter the certname of a node you want to add to the `iis` node group.
3. Click **Pin node**.
4. Repeat for additional nodes you want to add to the `iis` group.
5. When you are done pinning nodes, commit changes.

You have added your `iis` nodes into a node group. For more information on adding and classifying nodes, read about [Making changes to node groups](#) on page 326 and [Grouping and classifying nodes](#) on page 318.

Run Puppet on your IIS nodes

Run Puppet in the console to enforce your desired state on the `iis` node group you created.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Under **Run options**, do not select anything.
3. From the list of target types, select **Node group**.

4. In the **Chose a node group box**, search for the `iis` node group and click **Select**.
5. Click **Run job**.
View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun the it on all nodes or only on the nodes that failed during the initial run.

Congratulations! You have run Puppet on your `iis` node group. In the future, the IIS nodes will check in with PE every 30 minutes to confirm that their IIS configuration matches what you've specified, and PE will correct it if it doesn't.

Next, learn how to use roles and profiles to efficiently manage system configurations.

Organize webserver configurations with roles and profiles

The *roles and profiles* method is a reliable way to build reusable, configurable, and refactorable system configurations.

Roles and profiles help you pick out relevant pieces of code from modules and bundle them together to create your own custom set of code for managing things. *Profiles* are the actual bundles of code. *Roles* gather profiles together so that you can assign them to nodes. This allows you to efficiently organize your Puppet code.

To learn about roles and profiles by example, follow these instructions to define a profile that configures the `example.com` website and includes a firewall rule. Then, you will create a role to contain the profile, and you'll assign it to a the `iis` node group that you created earlier. This lays down a base structure where, if you had additional websites to serve, you would create additional profiles for them, and those profiles could be separated or combined inside the roles as needed.

Note: Adding a firewall rule isn't necessary for an IIS website because the port is already open, but the purpose of this example is to show that you can write a role that manages more than one piece of software to accomplish a task. The webserver role can manage both IIS and the firewall.

Because you are adding a firewall rule, make sure you add the `puppet/windows_firewall` module to your Puppetfile, following the same process you used to add the `iis` module. Remember to add the firewall modules dependencies — `puppetlabs/stdlib` and `puppetlabs/registry`.

```
mod 'puppet/windows_firewall', '2.0.2'
mod 'puppetlabs/stdlib' , '4.6.0'
mod 'puppetlabs/registry' , '1.1.1'
```

See [The roles and profiles method](#) on page 360 for more information about how roles and profiles work.

Set up your prerequisites

Before you begin writing content for roles and profiles, you need to create modules to store them in.

1. Create one module for `profile` and one for `role` directly in your control repo. Do not put them in your Puppetfile.
2. Make a new directory in the repo named `site`. For example, `/etc/puppetlabs/code/environments/production/site`.
3. Edit the `environment.conf` file to add `site` to the `modulepath`. This is the place where Puppet looks for module information. For example: `modulepath = site:modules:$basemodulepath`.
4. Put the `role` and `profile` modules in the `site` directory.

Write a profile for your IIS website

Write a webserver profile that includes rules for your `iis_site` and `firewall`.

Before you begin

Make sure you have already installed the `puppetlabs/iis` module and the `puppet/windows_firewall` module from the Forge.

Note: We recommend writing your code in a code editor, such as VSCode, and then pushing to your Git server. Puppet has an [extension](#) for VSCode that supports syntax highlighting of the Puppet language.

1. In the `profile` module you added, create the following directory:

- `manifests/`
 - `webserver/`
 - `example.pp`

2. Paste the following Puppet code into the new `example.pp` file:

```
class profile::webserver::example (
  String $content = 'Hello from iis',
  String $port = '80',
)
{
  windows_firewall::exception { 'http':
    ensure      => present,
    direction   => 'in',
    action      => 'allow',
    enabled     => true,
    protocol    => 'TCP',
    local_port  => Integer($port),
    remote_port => 'any',
    display_name => 'IIS incoming traffic HTTP-In',
    description => "Inbound rule for IIS web traffic. [TCP ${port}]",
  }

  $iis_features = ['Web-WebServer', 'Web-Scripting-Tools', 'Web-Mgmt-Console']
  iis_feature { $iis_features:
    ensure => 'present',
  }

  # Delete the default website to prevent a port binding conflict.
  iis_site {'Default Web Site':
    ensure => absent,
    require => Iis_feature['Web-WebServer'],
  }

  iis_site { 'minimal':
    ensure      => 'started',
    physicalpath => 'c:\\inetpub\\minimal',
    applicationpool => 'DefaultAppPool',
    bindings    => [
      {
        'bindinginformation' => "${facts['ipaddress']}:${port}:",
        'protocol'           => 'http',
      }
    ],
    require      => [
      File['minimal-index'],
      Iis_site['Default Web Site']
    ],
  }

  file { 'minimal':
    ensure => 'directory',
    path   => 'c:\\inetpub\\minimal',
  }

  file { 'minimal-index':
    ensure => 'file',
    path   => 'c:\\inetpub\\minimal\\index.html',
    content => $content,
    require => File['minimal']
  }
}
```

This profile applies custom rules for the `iis_site` class that include settings for `$port` and `$content`. The code uses `file` to ensure there is content on the main page of your site. Finally, there is a firewall rule that only allows traffic from the port or ports set in the `$port` setting.

You can add your own code to the profile as needed. Look at the [Readme](#) and [Reference](#) sections for the `puppetlabs/iis` and `puppet/windows_firewall` modules in the Forge for more content.

Set data for the profile

Hiera is a configuration method that allows you to set defaults in your code, or override those defaults in certain circumstances. Use it to fine-tune data within your profile.

Suppose you want to use the custom fact `stage` to represent the deployment stage of the node, which can be `dev`, `test`, or `prod`. For this example, use `dev` and `prod`.

With Hiera structured data, you can set up a four-layer hierarchy that consists of the following:

- `console_data` for data defined in the console.
- `nodes/%{trusted.certname}` for per-node overrides.
- `stage/%{facts.stage}` for setting stage-specific data.
- `common` for global fallback data.

This structure lets you tune the settings for ports and IPs in each stage.

For example, to configure webservers in the development environment to have a custom message and to use port 8080, you'd create a data file with the following name, location, and code content:

```
# /etc/puppetlabs/code/environments/production/data/stage/dev.yaml
---
profile::webserver::example::content: "Hello from dev"
profile::webserver::example::ports:
  - '8080'
```

To have webservers in the production environment listen to all interfaces:

```
# /etc/puppetlabs/code/environments/production/data/stage/prod.yaml
---
profile::webserver::example::ips:
  - '0.0.0.0'
  - ':::'
```

This is the briefest of introductions to all the things you can do with structured data in Hiera. To learn more about setting up hierarchical data, see [Getting started with Hiera](#).

Write a role for your IIS website

To write roles, think about what machines you'll be managing and decide what else they need in addition to the `webserver` profile.

Say you want all of the nodes in your `iis` node group to use the profile you just wrote. Suppose also that your organization assigns all machines, including workstations, with a profile called `profile::base`, which manages basic policies and uses some conditional logic to include operating-system-specific configuration.

Inside the following file in your control repo:

```
site-modules\role\manifests\exampleserver.pp
```

Write a role that includes both the base profile and your `webserver` profile:

```
class role::exampleserver {
  include profile::base
  include profile::webserver
}
```

You can add more profiles to this role, or create additional roles with more profile configurations based on your needs.

Assign the role to nodes

Assign the `exampleserver` role to nodes where you want to manage the configuration of the `iis_site`, based on what you wrote in the `webserver::example` profile.

For this example, assume you want to add `role::exampleserver` to all the nodes in the `iis` node group you created.

1. In the console, click **Node groups** and select the `iis` node group.
2. On the **Classes** tab, select `role::exampleserver` and click **Add class**.
3. Commit the change.

Your `iis` node group is managing your `iis_site` website based on the rules you coded into your `webserver` profile.

Next steps

Now that you have set up some basic automated configuration management with PE, here are some things to do next:

- Check out the [Forge](#) to download additional modules and start managing other things like [NTP](#) or machines running on [Azure](#).
- Learn how to develop high-quality modules with the [Puppet VSCode extension](#) and [Puppet Development Kit \(PDK\)](#).
- See the [Configuring Puppet Enterprise](#) on page 154 docs to fine-tune things like the console performance, orchestration services, Java, and proxy settings.
- Learn more about [Tasks in PE](#) on page 489 or [Plans in PE](#) on page 517.
- See [Managing access](#) on page 195 for information about adding and organizing other PE users and their permissions.
- If you have teams rolling out Puppet code changes across your infrastructure, check out [Getting Started with Continuous Delivery for Puppet Enterprise](#).
- Check out our [Youtube channel](#) to learn more about Puppet.

Installing

A Puppet Enterprise deployment typically includes infrastructure components and agents, which are installed on nodes in your environment.

You can install infrastructure components in multiple configurations and scale up with compilers. You can install agents on `*nix`, Windows, and macOS nodes.

- [Supported architectures](#) on page 77

There are several configurations available for Puppet Enterprise. The configuration you use depends on the number of nodes in your environment and the resources required to serve agent catalogs. All PE installations begin with the standard configuration, and scale up by adding additional components as needed.

- [System requirements](#) on page 81

Refer to these system requirements for Puppet Enterprise installations.

- [What gets installed and where?](#) on page 97

Puppet Enterprise installs several software components, configuration files, databases, services and users, and log files. It's useful to know the locations of these should you ever need to troubleshoot or manage your infrastructure.

- [Installing Puppet Enterprise](#) on page 102

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

- [Purchasing and installing a license key](#) on page 111

Your license must support the number of nodes that you want to manage with Puppet Enterprise.

- [Installing agents](#) on page 112

You can install Puppet Enterprise agents on *nix, Windows, and macOS.

- [Installing compilers](#) on page 132

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

- [Installing PE client tools](#) on page 138

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

- [Uninstalling](#) on page 142

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

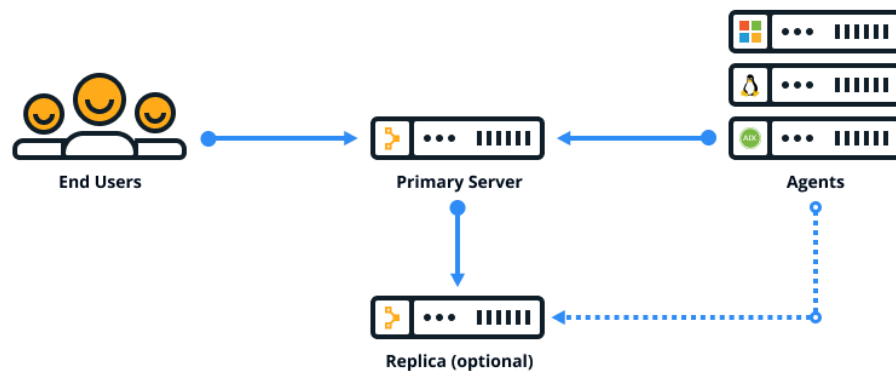
Supported architectures

There are several configurations available for Puppet Enterprise. The configuration you use depends on the number of nodes in your environment and the resources required to serve agent catalogs. All PE installations begin with the standard configuration, and scale up by adding additional components as needed.

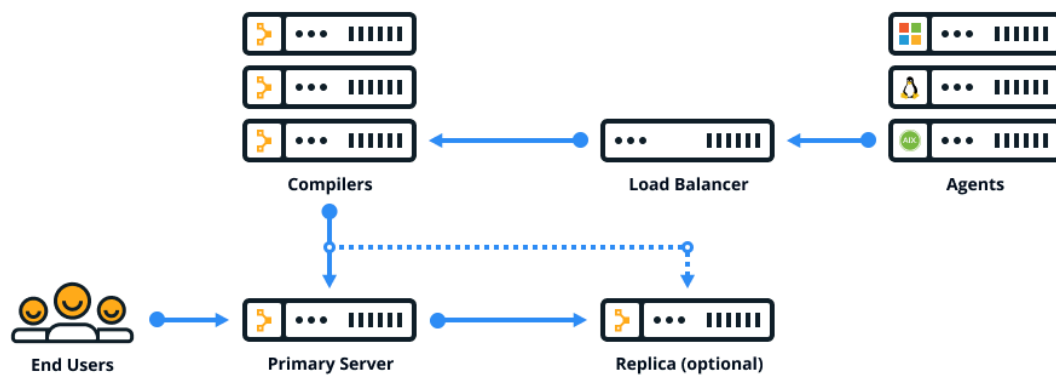
Existing customer deployments in the field may use other configurations, but for the best performance, scalability, and support, we recommend using one of these three defined architectures unless specifically advised otherwise by Puppet Support personnel. For legacy architectures, we document only upgrade procedures – not installation instructions – in order to support existing customers.

Configuration	Description	Node limit
Standard installation (Recommended)	All infrastructure components are installed on the primary server. This installation type is the easiest to install, upgrade, and troubleshoot.	Up to 4,000
Large installation	Similar to a standard installation, plus one or more compilers and a load balancer which help distribute the agent catalog compilation workload.	4,000–20,000
Extra-large installation	Similar to a large installation, plus one or more separate PE-PostgreSQL nodes that run PuppetDB.	More than 20,000

Standard installation

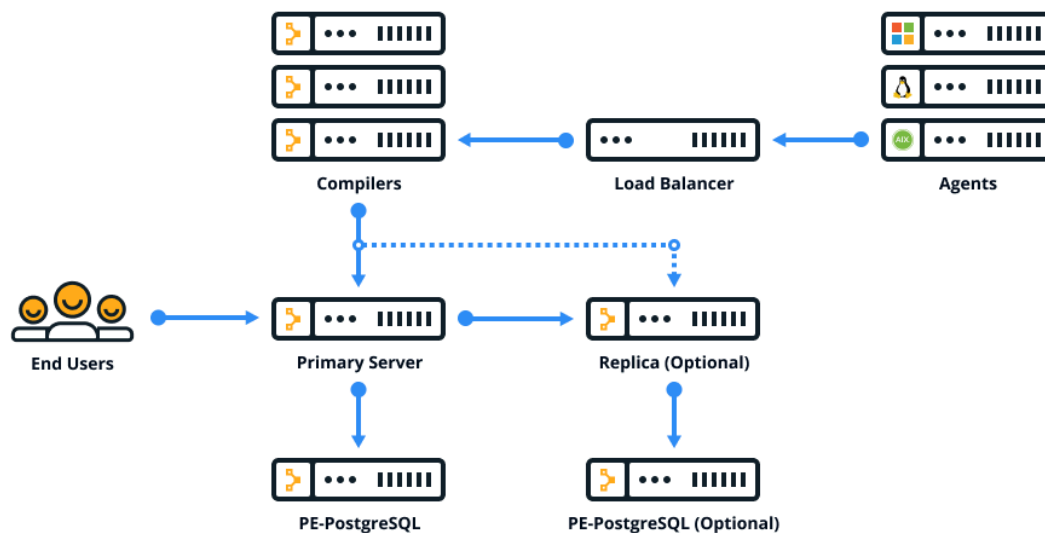


Large installation



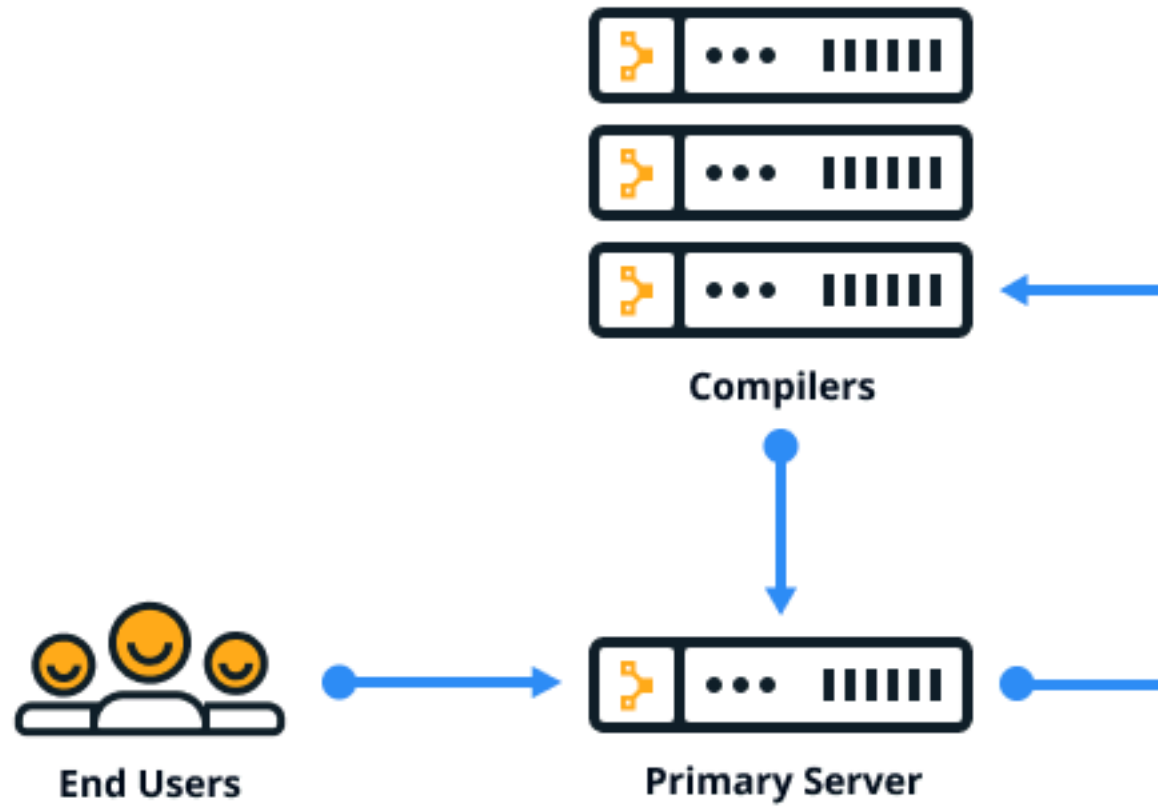
Extra-large installation

The extra-large architecture scales PE deployments to 20,000+ nodes. This architecture is intended to be deployed with the help of Puppet solutions experts. For more information about the capabilities of this architecture and how to deploy it, reach out to your technical account manager.



Standalone PE-PostgreSQL (legacy)

Upgrading from the retired split architecture results in a standalone PE-PostgreSQL architecture. This architecture is similar to a large installation, but with a separate node that hosts the PE-PostgreSQL instance. Standalone PE-PostgreSQL can't be configured with disaster recovery.



System requirements

Refer to these system requirements for Puppet Enterprise installations.

- [Hardware requirements](#) on page 81

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

- [Supported operating systems](#) on page 82

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

- [Supported browsers](#) on page 87

The following browsers are supported for use with the console.

- [System configuration](#) on page 87

Before installing Puppet Enterprise, make sure that your nodes and network are properly configured.

Hardware requirements

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

Your configuration and code base can significantly affect performance. Use PE tuning and metrics tools to further customize and refine your installation.

Tip:

If possible, address performance limitations by maximizing your hardware first, then scaling up to the next size architecture as needed. It's often easier to upgrade your hardware than to add additional infrastructure nodes.

Related information

[Tuning infrastructure nodes](#) on page 154

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

Hardware requirements for standard installations

These are the minimum hardware requirements for the primary server in a standard architecture with up to 2,500 nodes.

Node volume	Cores	RAM	/opt/	/var/	AWS EC2	Azure
Trial use	2	8 GB	20 GB	24 GB	m5.large	D2 v4
11–100	6	10 GB	50 GB	24 GB	c5.2xlarge	F8s v2
101–500	8	12 GB	50 GB	24 GB	c5.2xlarge	F8s v2
501–1,000	10	16 GB	50 GB	24 GB	c5.2xlarge	F8s v2
1,000–2,500	12	24 GB	50 GB	24 GB	c5.4xlarge	F16s v2

Notes on the data in this chart:

- **Trial mode:** Although the m5.large instance type is sufficient for trial use, it is not supported. A minimum of four cores is required for production workloads.
- **Azure:** Azure requirements are not currently tested by Puppet, but are presented here as our best guidance based on comparable EC2 instance testing.
- **/opt/ storage requirements:** The database should not exceed 50% of /opt/ to allow for future upgrades.
- **/var/ storage requirements:** There are roughly 20 log files stored in /var/ which are limited in size to 1 GB each. We recommend allocating 24 GB to avoid issues, however log retention settings generally prevent reaching the maximum capacity.

Hardware requirements for large installations

These are the minimum hardware requirements for the primary server and compilers in a large architecture with 2,500–20,000 nodes.

Each compiler increases capacity by approximately 1,500–3,000 nodes, until you exhaust the capacity of PuppetDB or the console, which run on the primary server.

Node volume	Node	Cores	RAM	/opt/	/var/	EC2
2,500–20,000	Primary node	16	32 GB	150 GB	10 GB	c5.4xlarge
	Each compiler (1,500 - 3,000 nodes)	6	12 GB	30 GB	2 GB	m5.xlarge

Hardware requirements for extra-large installations

These are the minimum hardware requirements for the primary server, compilers, and PE-PostgreSQL nodes in an extra-large architecture with 20,000+ nodes.

Node volume	Node	Cores	RAM	/opt/	/var/	EC2
20,000+	Primary node	16	32 GB	150 GB	10 GB	c5.4xlarge
	Each compiler (1,500 - 3,000 nodes)	6	12 GB	30 GB	2 GB	m5.xlarge
	PE-PostgreSQL node	16	128 GB	300 GB	4 GB	r5.4xlarge

If you manage more than 20,000 nodes, contact Puppet professional services to talk about optimizing your setup for your specific requirements

Supported operating systems

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

Supported operating systems and devices

When choosing an operating system, first consider the machine's role. Different roles support different operating systems and architectures.

Tip: For details about platform support lifecycles and planned end-of-life support, see [Platform support lifecycle](#) on the Puppet website.

Primary server platforms

The primary server can be installed on these operating systems and architectures.

Operating system	Versions	Architecture
Enterprise Linux	7, 8	<ul style="list-style-type: none"> x86_64 FIPS 140-2 compliant RHEL version 7
<ul style="list-style-type: none"> CentOS Oracle Linux Red Hat Enterprise Linux Scientific Linux 		

Operating system	Versions	Architecture
SUSE Linux Enterprise Server	12	x86_64
Ubuntu (General Availability kernels)	18.04	amd64

Agent platforms

The agent role can be installed on these operating systems and architectures.

Operating system	Versions	Architecture
AIX	7.1, 7.2 Note: We support only technology levels that are still under support from IBM.	
Amazon Linux	2	
Debian	Stretch (9), Buster (10)	<ul style="list-style-type: none"> i386 for version 9 amd64
Enterprise Linux <ul style="list-style-type: none"> CentOS Oracle Linux Red Hat Enterprise Linux Scientific Linux 	6, 7, 8 Note: Scientific Linux 8 is not supported.	<ul style="list-style-type: none"> x86_64 i386 for versions 6 aarch64 for version 7, 8 FIPS 140-2 compliant RHEL version 7
Fedora	30, 31	<ul style="list-style-type: none"> x86_64
macOS	10.14, 10.15	
Microsoft Windows	10	<ul style="list-style-type: none"> x64 x86 FIPS 140-2 compliant x64 version 10
Microsoft Windows Server	2012, 2012 R2, 2012 R2 core, 2016, 2016 core, 2019, 2019 core	<ul style="list-style-type: none"> x64 for 2012 versions FIPS 140-2 compliant x64 versions 2012 R2 and 2012 R2 core
Solaris	11	<ul style="list-style-type: none"> SPARC i386
SUSE Linux Enterprise Server	12, 15	<ul style="list-style-type: none"> x86_64
Ubuntu (General Availability kernels)	16.04, 18.04, 20.04	<ul style="list-style-type: none"> amd64 i386 for versions 16.04

Note: Some operating systems require an active subscription with the vendor's package management system (for example, the Red Hat Network) to install dependencies.

CentOS dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

	All nodes	Primary server
cronie	x	x
dmidecode	x	x
libxml2	x	x
logrotate	x	x
net-tools	x	x
pciutils	x	x
which	x	x
zlib	x	x
curl		x
libjpeg		x
libtool-ltdl (versions 7 and later)		x
libxslt		x
mailcap		x

RHEL dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

	All nodes	Primary server
cronie	x	x
dmidecode	x	x
libxml2	x	x
logrotate	x	x
net-tools	x	x
pciutils	x	x
which	x	x
zlib	x	x
curl		x
libjpeg		x
libtool-ltdl (versions 7 and later)		x
libxslt		x
mailcap		x

SUSE Linux Enterprise Server dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, most dependencies are installed during installation. If you encounter problems, inspect the error messages for packages that require other SUSE Linux Enterprise Server packaging modules to be enabled, and use `zypper package-search <PACKAGE NAME>` to locate them for manual installation.

	All nodes	Primary server
cron	x	x
libxml2	x	x
libxslt	x	x
logrotate	x	x
net-tools	x	x
pciutils	x	x
pmtools	x	x
zlib	x	x
curl		x
db43		x
libjpeg		x
unixODBC		x

Ubuntu dependencies

When you install PE or an agent, these packages are also installed from the various operating system repositories.

If the machine you're installing on has internet access, dependencies are set up during installation. If your machine doesn't have internet access, you must manually install dependencies.

	All nodes	Primary server
cron	x	x
dmidecode	x	x
gnupg	x	x
hostname	x	x
libldap-2.4-2	x	x
libreadline5	x	x
libxml2	x	x
logrotate	x	x
pciutils	x	x
zlib	x	x
ca-certificates-java		x
curl		x
file		x
libcap2		x

All nodes	Primary server
libgtk2.0-0	x
libjpeg62	x
libmagic1	x
libssp-uuid16	x
libpcre3	x
libxslt1.1	x
mime-support	x
perl	x

AIX dependencies and limitations

Before installing the agent on AIX systems, install these packages.

- bash
- zlib
- readline
- curl
- openssl



CAUTION: For curl and openssl packages, you must use the versions provided by the "AIX Toolbox Cryptographic Content" repository, which is available via IBM support. Note that the cURL version must be 7.9.3. Do not use the cURL version in the AIX toolbox package for Linux applications, as that version does not include support for OpenSSL.

To install the bash, zlib, and readline packages on a node directly, run `rpm -Uvh` with the following URLs. The RPM package provider must be run as root.

- <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/bash/bash-3.2-1.aix5.2.ppc.rpm>
- <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/zlib/zlib-1.2.3-4.aix5.2.ppc.rpm>
- <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/readline/readline-6.1-1.aix6.1.ppc.rpm> (AIX 6.1 and 7.1 only)
- <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/readline/readline-4.3-2.aix5.1.ppc.rpm> (AIX 5.3 only)

If you are behind a firewall or running an http proxy, the above commands might not work. Instead, use the AIX toolbox packages download available from IBM.

GPG verification does not work on AIX, because the RPM version it uses is too old. The AIX package provider doesn't support package downgrades (installing an older package over a newer package). Avoid using leading zeros when specifying a version number for the AIX provide, for example, use `2.3.4` not `02.03.04`.

The PE AIX implementation supports the NIM, BFF, and RPM package providers. Check the type reference for technical details on these providers.

Upgrade your operating system with PE installed

If you have PE installed, take extra precautions before performing a major upgrade of your machine's operating system.

Performing major upgrades of your operating system with PE installed can cause errors and issues with PE. A major operating system upgrade is an upgrade to a new whole version, such as an upgrade from CentOS 6.0 to 7.0; it does not refer to a minor version upgrade, like CentOS 6.5 to 6.6. Major upgrades typically require a new version of PE.

1. Back up your databases and other PE files.
2. Perform a complete uninstall (using the `-p` and `-d` uninstaller options).
3. Upgrade your operating system.

4. Install PE.
5. Restore your backup.

Related information

[Back up your infrastructure](#) on page 687

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

[Restore your infrastructure](#) on page 688

Use the restore commands to migrate your primary server to a new host or to recover from system failure.

[Uninstalling](#) on page 142

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

[Installing Puppet Enterprise](#) on page 102

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

Supported browsers

The following browsers are supported for use with the console.

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	10 or later

System configuration

Before installing Puppet Enterprise, make sure that your nodes and network are properly configured.

Note: Port numbers are Transmission Control Protocols (TCP), unless noted otherwise.

Network considerations

Before installing, consider these network requirements

Timekeeping

Use NTP or an equivalent service to ensure that time is in sync between your primary server, which acts as the certificate authority, and any agent nodes. If time drifts out of sync in your infrastructure, you might encounter issues such as agents receiving outdated certificates. A service like NTP (available as a supported module) ensures accurate timekeeping.

Name resolution

Decide on a preferred name or set of names that agent nodes can use to contact the primary server. Ensure that the primary server can be reached by domain name lookup by all future agent nodes.

You can simplify configuration of agent nodes by using a CNAME record to make the primary server reachable at the hostname `puppet`, which is the default primary server hostname that is suggested when installing an agent node.

Web URLs used for deployment and management

PE uses some external web URLs for certain deployment and management tasks. You might want to ensure these URLs are reachable from your network prior to installation, and be aware that they might be called at various stages of configuration.

URL	Enables
forgeapi.puppet.com	Puppet module downloads.
pm.puppetlabs.com	Agent module package downloads.
s3.amazonaws.com	Agent module package downloads (redirected from pm.puppetlabs.com).
rubygems.org	Puppet and Puppet Server gem downloads.
github.com	Third-party module downloads not served by the Forge and access to control repositories.

Antivirus and antimalware considerations

Antivirus and antimalware software can impact or prevent the proper functioning of PE. To avoid issues, exclude the directories `/etc/puppetlabs` and `/opt/puppetlabs` from antivirus and antimalware tools that scan disk write operations.

- Exclude the `/etc/puppetlabs` and `/opt/puppetlabs` directories from antivirus and antimalware tools that scan disk write operations to avoid performance issues.
- Some antivirus and antimalware software requires a lot of system processing power. Tune your system resources to accommodate the software so it doesn't slow your performance.
- Some antivirus and antimalware software defaults to using port 8081, which is the same port PuppetDB uses. When installing the software, consider which port it uses so it doesn't conflict with PuppetDB communications.

Related information

[Tuning infrastructure nodes](#) on page 154

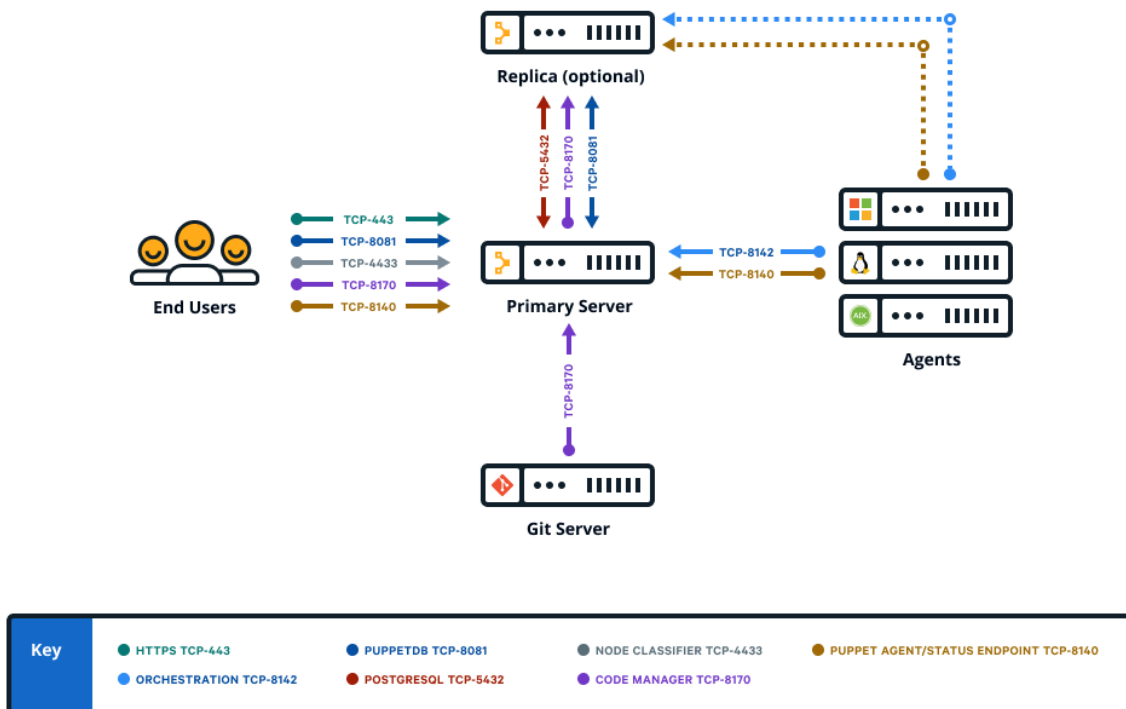
Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

[The default port for PuppetDB conflicts with another service](#) on page 700

By default, PuppetDB communicates over port 8081. In some cases, this might conflict with existing services, for example McAfee ePolicy Orchestrator.

Firewall configuration for standard installations

These are the port requirements for standard installations.

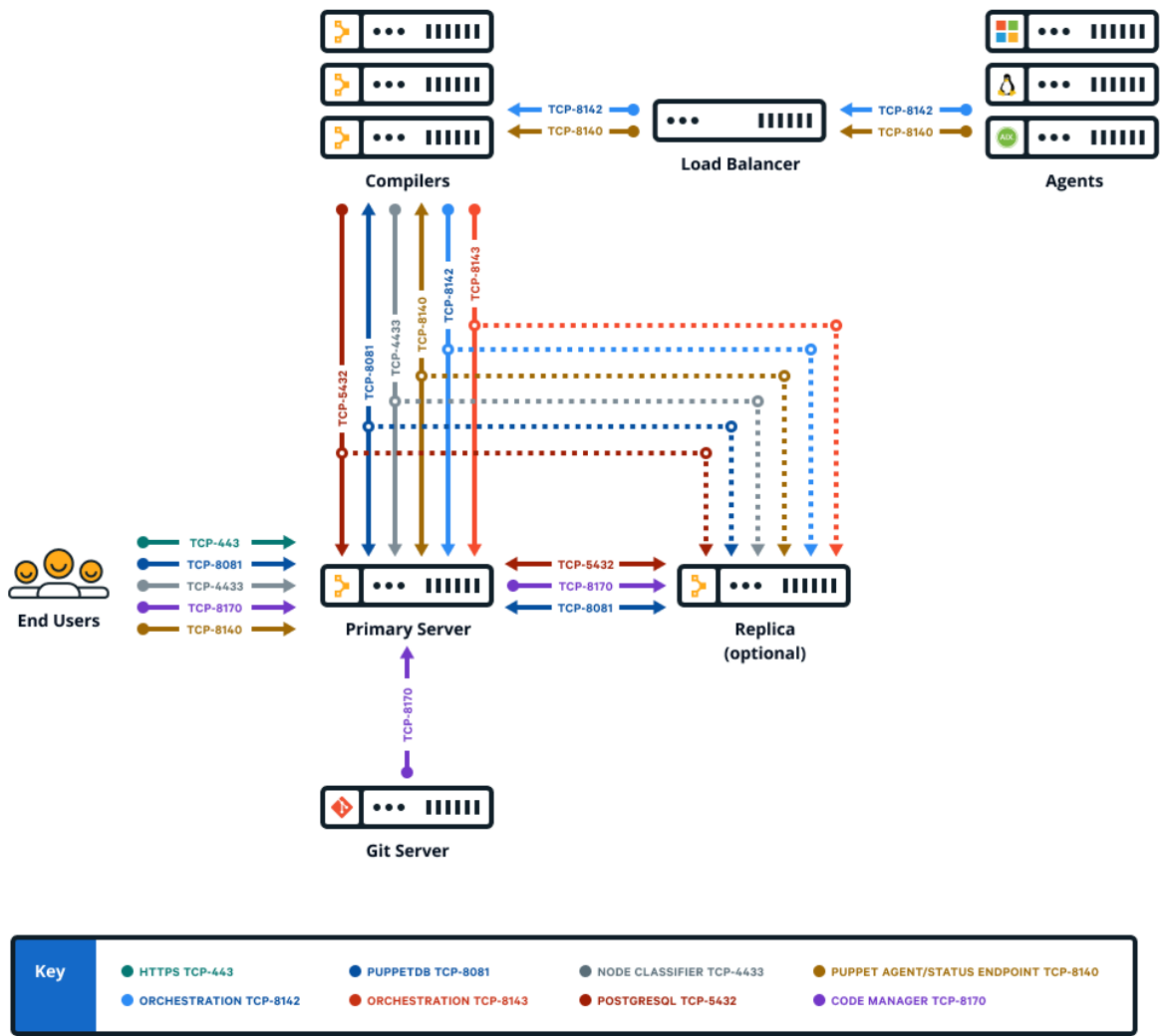


Port	Use
443	<ul style="list-style-type: none"> This port provides host access to the console The console accepts HTTPS traffic from end users on this port. Classifier group: PE Console
4433	<ul style="list-style-type: none"> This port is used as a classifier / console services API endpoint. The primary server communicates with the console over this port. Classifier group: PE Console
5432	<ul style="list-style-type: none"> This port is used to replicate PostgreSQL data between the primary server and replica.

Port	Use
8081	<ul style="list-style-type: none"> • PuppetDB accepts traffic/requests on this port. • The primary server and console send traffic to PuppetDB on this port. • PuppetDB status checks are sent over this port. • Classifier group: PE PuppetDB
8140	<ul style="list-style-type: none"> • The primary server uses this port to accept inbound traffic/requests from agents. • The console sends requests to the primary server on this port. • Certificate requests are passed over this port unless <code>ca_port</code> is set differently. • Puppet Server status checks are sent over this port. • Classifier group: PE Master
8142	<ul style="list-style-type: none"> • Orchestrator and the Run Puppet button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent. • Classifier group: PE Orchestrator
8170	<ul style="list-style-type: none"> • Code Manager uses this port to deploy environments, run webhooks, and make API calls.

Firewall configuration for large installations

These are the port requirements for large installations with compilers.

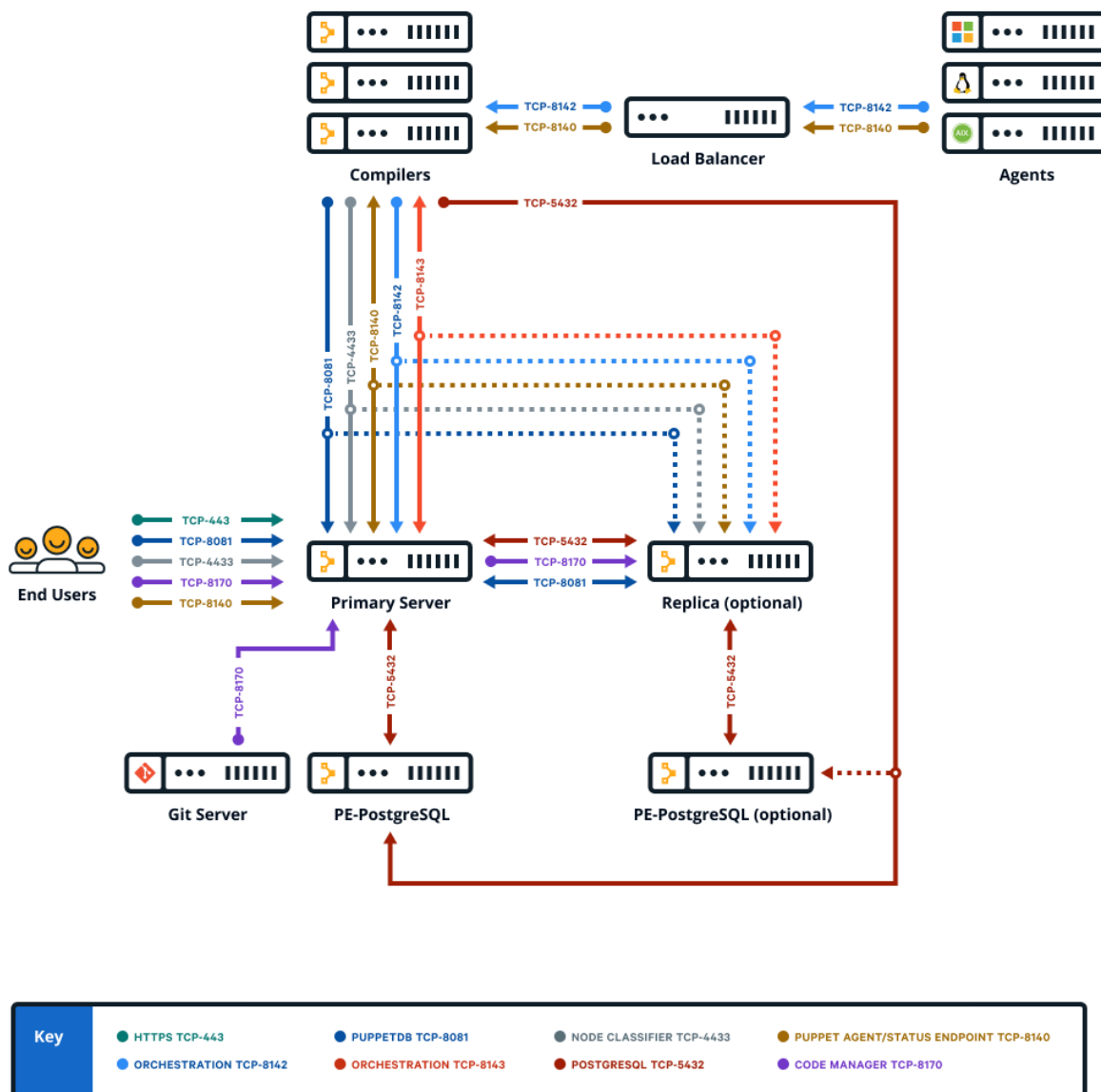


Port	Use
443	<ul style="list-style-type: none">This port provides host access to the consoleThe console accepts HTTPS traffic from end users on this port.Classifier group: PE Console
4433	<ul style="list-style-type: none">This port is used as a classifier / console services API endpoint.The primary server communicates with the console over this port.Classifier group: PE Console

Port	Use
5432	<ul style="list-style-type: none"> This port is used to replicate PostgreSQL data between the primary server and replica. This port is used by the PuppetDB service running on compilers to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic/requests on this port. The primary server and console send traffic to PuppetDB on this port. PuppetDB status checks are sent over this port. Classifier group: PE PuppetDB
8140	<ul style="list-style-type: none"> The primary server uses this port to accept inbound traffic/requests from agents. The console sends requests to the primary server on this port. Certificate requests are passed over this port unless <code>ca_port</code> is set differently. Puppet Server status checks are sent over this port. The primary server uses this port to send status checks to compilers. (Not required to run PE.) Classifier group: PE Master
8142	<ul style="list-style-type: none"> Orchestrator and the Run Puppet button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent. Classifier group: PE Orchestrator
8143	<ul style="list-style-type: none"> Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation. Classifier group: PE Orchestrator
8170	<ul style="list-style-type: none"> Code Manager uses this port to deploy environments, run webhooks, and make API calls.

Firewall configuration for extra-large installations

These are the port requirements for extra-large installations with compilers.



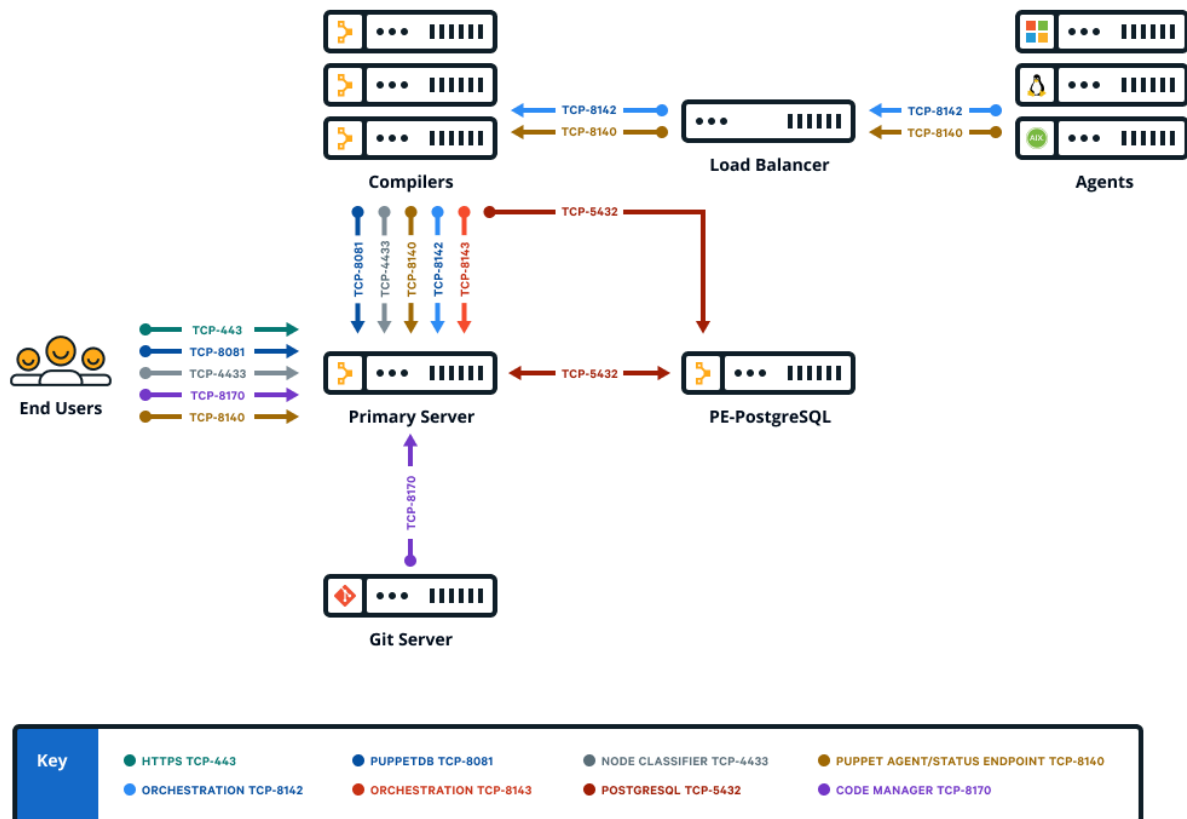
Port	Use
443	<ul style="list-style-type: none"> This port provides host access to the console The console accepts HTTPS traffic from end users on this port. Classifier group: PE Console

Port	Use
4433	<ul style="list-style-type: none"> This port is used as a classifier / console services API endpoint. The primary server communicates with the console over this port. Classifier group: PE Console
5432	<ul style="list-style-type: none"> This port is used to replicate PostgreSQL data between the primary server and replica. This port is used by the PuppetDB service running on compilers to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic/requests on this port. The primary server and console send traffic to PuppetDB on this port. PuppetDB status checks are sent over this port. Classifier group: PE PuppetDB
8140	<ul style="list-style-type: none"> The primary server uses this port to accept inbound traffic/requests from agents. The console sends requests to the primary server on this port. Certificate requests are passed over this port unless <code>ca_port</code> is set differently. Puppet Server status checks are sent over this port. The primary server uses this port to send status checks to compilers. (Not required to run PE.) Classifier group: PE Master
8142	<ul style="list-style-type: none"> Orchestrator and the Run Puppet button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent. Classifier group: PE Orchestrator

Port	Use
8143	<ul style="list-style-type: none"> Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation. Classifier group: PE Orchestrator
8170	<ul style="list-style-type: none"> Code Manager uses this port to deploy environments, run webhooks, and make API calls.

Firewall configuration for standalone PE-PostgreSQL installations

These are the port requirements for installations with compilers and standalone PE-PostgreSQL



Port	Use
443	<ul style="list-style-type: none"> • This port provides host access to the console • The console accepts HTTPS traffic from end users on this port. • Classifier group: PE Console
4433	<ul style="list-style-type: none"> • This port is used as a classifier / console services API endpoint. • The primary server communicates with the console over this port. • Classifier group: PE Console
5432	<ul style="list-style-type: none"> • The standalone PE-PostgreSQL node uses this port to accept inbound traffic/requests from the primary server. • This port is used by the PuppetDB service running on compilers to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> • PuppetDB accepts traffic/requests on this port. • The primary server and console send traffic to PuppetDB on this port. • PuppetDB status checks are sent over this port. • Classifier group: PE PuppetDB
8140	<ul style="list-style-type: none"> • The primary server uses this port to accept inbound traffic/requests from agents. • The console sends requests to the primary server on this port. • Certificate requests are passed over this port unless <code>ca_port</code> is set differently. • Puppet Server status checks are sent over this port. • The primary server uses this port to send status checks to compilers. (Not required to run PE.) • Classifier group: PE Master

Port	Use
8142	<ul style="list-style-type: none"> Orchestrator and the Run Puppet button use this port on the primary server to accept inbound traffic/responses from agents via the Puppet Execution Protocol agent. Classifier group: PE Orchestrator
8143	<ul style="list-style-type: none"> Orchestrator uses this port to accept connections from Puppet Communications Protocol brokers to relay communications. The orchestrator client also uses this port to communicate with the orchestration services running on the primary server. If you install the orchestrator client on a workstation, port 8143 on the primary server must be accessible from the workstation. Classifier group: PE Orchestrator
8170	<ul style="list-style-type: none"> Code Manager uses this port to deploy environments, run webhooks, and make API calls.

What gets installed and where?

Puppet Enterprise installs several software components, configuration files, databases, services and users, and log files. It's useful to know the locations of these should you ever need to troubleshoot or manage your infrastructure.

Software components installed

PE installs several software components and dependencies. These tables show which version of each component is installed for releases dating back to the previous long term supported (LTS) release.

The functional components of the software are separated between those packaged with the agent and those packaged on the server side (which also includes the agent).

Note: PE also installs other dependencies, as documented in the system requirements.

This table shows the components installed on all agent nodes.

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

PE Version	Puppet and the Puppet agent	Facter	Hiera	Ruby	OpenSSL
2021.0	7.4.1	4.0.51	3.6.0	2.7.2	1.1.1i
2019.8.5 (LTS)	6.21.1	3.14.16	3.6.0	2.5.8	1.1.1i
2019.8.4	6.19.1	3.14.14	3.6.0	2.5.8	1.1.1g
2019.8.3	6.19.1	3.14.14	3.6.0	2.5.8	1.1.1g
2019.8.1	6.17.0	3.14.12	3.6.0	2.5.8	1.1.1g
2019.8	6.16.0	3.14.11	3.6.0	2.5.8	1.1.1g

This table shows components installed on server nodes.

PE Version	Puppet Server	PuppetDB	r10k	Bolt Services	Agentless Catalog Executor (ACE) Services	PostgreSQL	Java	Nginx
2021.0	7.0.3	7.1.0	3.8.0	3.0.0	1.2.2	11.10	11.0	1.19.6
2019.8.5 (LTS)	6.15.1	6.14.0	3.8.0	3.0.0	1.2.2	11.10	11.0	1.19.6
2019.8.4	6.14.1	6.13.1	3.6.0	2.32.0	1.2.1	11.10	11.0	1.17.10
2019.8.3	6.14.1	6.13.1	3.6.0	2.32.0	1.2.1	11.9	11.0	1.17.10
2019.8.1	6.12.1	6.11.3	3.5.2	2.16.0	1.2.0	11.8	11.0	1.17.10
2019.8	6.12.0	6.11.1	3.5.1	2.11.1	1.2.0	11.8	11.0	1.17.10

Executable binaries and symlinks installed

PE installs executable binaries and symlinks for interacting with tools and services.

On *nix nodes, all software is installed under `/opt/puppetlabs`.

On Windows nodes, all software is installed in Program Files at `Puppet Labs\Puppet`.

Executable binaries on *nix are in `/opt/puppetlabs/bin` and `/opt/puppetlabs/sbin`.

Tip: To include binaries in your default `$PATH`, manually add them to your profile or export the path:

```
export PATH=$PATH:/opt/puppetlabs/bin
```

To make essential Puppet tools available to all users, the installer automatically creates symlinks in `/usr/local/bin` for the `facter`, `puppet`, `pe-man`, `r10k`, and `hiera` binaries. Symlinks are created only if `/usr/local/bin` is writeable. Users of AIX and Solaris versions 10 and 11 must add `/usr/local/bin` to their default path.

For macOS agents, symlinks aren't created until the first successful run that applies the agents' catalogs.

Tip: You can disable symlinks by changing the `manage_symlinks` setting in your default Hieradata file:

```
puppet_enterprise::manage_symlinks: false
```

Binaries provided by other software components, such as those for interacting with the PostgreSQL server, PuppetDB, or Ruby packages, do not have symlinks created.

Modules and plugins installed

PE installs modules and plugins for normal operations.

Modules included with the software are installed on the primary server in `/opt/puppetlabs/puppet/modules`. Don't modify anything in this directory or add modules of your own. Instead, install non-default modules in `/etc/puppetlabs/code/environments/<environment>/modules`.

Configuration files installed

PE installs configuration files that you might need to interact with from time to time.

On *nix nodes, configuration files live at `/etc/puppetlabs`.

On Windows nodes, configuration files live at `<COMMON_APPDATA>\PuppetLabs`. The location of this folder varies by Windows version; in 2008 and 2012, its default location is `C:\ProgramData\PuppetLabs\puppet\etc`.

The agent software's `confdir` is in the `puppet` subdirectory. This directory contains the `puppet.conf` file, `auth.conf`, and the SSL directory.

Tools installed

PE installs several suites of tools to help you work with the major components of the software.

- **Puppet tools** — Tools that control basic functions of the software such as `puppet agent` and `puppet ssl`.
- **Puppet Server tools** — The primary server contains a tool to manage and interact with the provided certificate authority, `puppetserver ca`.
- **Client tools** — The `pe-client-tools` package collects a set of CLI tools that extend the ability for you to access services from the primary server or a workstation. This package includes:
 - **Orchestrator** — The orchestrator is a set of interactive command line tools that provide the interface to the orchestration service. Orchestrator also enables you to enforce change on the environment level. Tools include `puppet job` and `puppet app`.
 - **Puppet Access** — Users can generate tokens to authenticate their access to certain command line tools and API endpoints.
 - **Code Manager CLI** — The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.
 - **PuppetDB CLI** — This a tool for working with PuppetDB, including building queries and handling exports.
- **Module tool** — The module tool is used to access and create modules, which are reusable chunks of Puppet code users have written to automate configuration and deployment tasks. For more information, and to access modules, visit the Forge.
- **Console** — The console is the web user interface for PE. The console provides tools to view and edit resources on your nodes, view reports and activity graphs, and more.

Databases installed

PE installs several default databases, all of which use PostgreSQL as a database backend.

The PE PostgreSQL database includes these following databases.

Database	Description
pe-activity	Activity data from the classifier, including who, what, and when.
pe-classifier	Classification data, all node group information.
pe-puppetdb	PuppetDB data, including exported resources, catalogs, facts, and reports.
pe-rbac	RBAC data, including users, permissions, and AD/LDAP info.
pe-orchestrator	Orchestrator data, including details about job runs.

Use the native PostgreSQL tools to perform database exports and imports. At a minimum, perform backups to a remote system nightly, or as dictated by your company policy.

Services installed

PE installs several services used to interact with the software during normal operations.

Service	Definition
pe-console-services	Manages and serves the console.
pe-puppetserver	Runs the primary server.
pe-nginx	Nginx, serves as a reverse-proxy to the console.
puppet	(on Enterprise Linux and Debian-based platforms) Runs the agent daemon on every agent node.

Service	Definition
pe-puppetdb, pe-postgresql	Daemons that manage and serve the database components. The pe-postgresql service is created only if the software installs and manages PostgreSQL.
pxp-agent	Runs the Puppet Execution Protocol agent process.
pe-orchestration-services	Runs the orchestration process.
pe-ace-server	Runs the ace server.
pe-bolt-server	Runs the Bolt server.

User and group accounts installed

These are the user and group accounts installed.

User	Definition
pe-puppet	Runs the primary server processes spawned by pe-puppetserver.
pe-webserver	Runs Nginx.
pe-puppetdb	Has root access to the database.
pe-postgres	Has access to the pe-postgreSQL instance. Created only if the software installs and manages PostgreSQL.
pe-console-services	Runs the console process.
pe-orchestration-services	Runs the orchestration process.
pe-ace-server	Runs the ace server.
pe-bolt-server	Runs the Bolt server.

Log files installed

The software distributed with PE generates log files that you can collect for compliance or use for troubleshooting.

Primary server logs

The primary server has these logs.

- `/var/log/puppetlabs/puppetserver/code-manager-access.log`
- `/var/log/puppetlabs/puppetserver/file-sync-access.log`
- `/var/log/puppetlabs/puppetserver/masterhttp.log`
- `/var/log/puppetlabs/puppetserver/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on compilers.
- `/var/log/puppetlabs/puppetserver/puppetserver.log` — The primary server logs its activity, including compilation errors and deprecation warnings, here.
- `/var/log/puppetlabs/puppetserver/puppetserver-access.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/puppetserver/puppetserver-status.log`

Agent logs

The locations of agent logs depend on the agent operating system.

On *nix nodes, the agent service logs its activity to the syslog service. Your syslog configuration dictates where these messages are saved, but the default location is `/var/log/messages` on Linux, `/var/log/system.log` on macOS, and `/var/adm/messages` on Solaris.

On Windows nodes, the agent service logs its activity to the event log. You can view its logs by browsing the event viewer.

Console and console services logs

The console and pe-console-services has these logs.

- `/var/log/puppetlabs/console-services/console-services.log`
- `/var/log/puppetlabs/console-services/console-services-api-access.log`
- `/var/log/puppetlabs/console-services-access.log`
- `/var/log/puppetlabs/console-services-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/nginx/access.log`
- `/var/log/puppetlabs/nginx/error.log` — Contains errors related to nginx. Console errors that aren't logged elsewhere can be found in this log.

Installer logs

The installer has these logs.

- `/var/log/puppetlabs/installer/http.log` — Contains web requests sent to the installer. This log is present only on the machine from which a web-based installation was performed.
- `/var/log/puppetlabs/installer/orchestrator_info.log` — Contains run details about puppet infrastructure commands that use the orchestrator, including the commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.
- `/var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log` — Contains the contents of the last installer run.
- `/var/log/puppetlabs/installer/installer-<timestamp>.log` — Contains the operations performed and any errors that occurred during installation.
- `/var/log/puppetlabs/installer/<action_name>_<timestamp>_<run_description>.log` — Contains details about disaster recovery command execution. Each action triggers multiple Puppet runs, some on the primary server, some on the replica, so there might be multiple log files for each command.

Database logs

The database has these logs.

- `/var/log/puppetlabs/postgresql/9.6/pgstartup.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Mon.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Tue.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Wed.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Thu.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Fri.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Sat.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Sun.log`
- `/var/log/puppetlabs/puppetdb/puppetdb.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-access.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-status.log`

Orchestration logs

The orchestration service and related components have these logs.

- `/var/log/puppetlabs/orchestration-services/aggregate-node-count.log`
- `/var/log/puppetlabs/orchestration-services/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on the primary server.
- `/var/log/puppetlabs/orchestration-services/pcp-broker-access.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services-access.log`
- `/var/log/puppetlabs/orchestration-services/orchestration-services-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/orchestration-services/orchestration-services-status.log`
- `/var/log/puppetlabs/pxp-agent/pxp-agent.log` (on *nix) or `C:\ProgramData\PuppetLabs/pxp-agent/var/log/pxp-agent.log` (on Windows) — Contains the Puppet Execution Protocol agent log file.
- `/var/log/puppetlabs/bolt-server/bolt-server.log` - Log file for PE Bolt server.
- `/var/log/puppetlabs/orchestration-services/orchestration-services.log` — Log file for the node inventory service.

Certificates installed

During installation, the software generates and installs a number of SSL certificates so that agents and services can authenticate themselves.

These certs can be found at `/etc/puppetlabs/puppet/ssl/certs`.

A certificate with the same name as the agent that runs on the primary server is generated during installation. This certificate is used by PuppetDB and the console.

Services that run on the primary server — for example, `pe-orchestration-services` and `pe-console-services` — use the agent certificate to authenticate.

The certificate authority, if active, stores its certificate information at `/etc/puppetlabs/puppetserver/ca`.

Secret key file installed

During installation, the software generates a secret key file that is used to encrypt and decrypt sensitive data stored in the inventory service.

The secret key is stored at: `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`

Installing Puppet Enterprise

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

To install a FIPS-enabled PE primary server, install the appropriate FIPS-enabled PE tarball, for example `puppet-enterprise-2021.0.0-redhatfips-7-x86_64.tar`, on a third party [supported platform](#) with FIPS mode enabled. The node must be configured with sufficient available entropy for the installation process to succeed.

Install PE

Installation uses default settings to install all of the PE infrastructure components on a single node. After installing, you can scale or customize your installation as needed.

Important: Perform these steps on your target primary server logged in as root. If you're installing on a system that doesn't enable root login, switch to the root user with this command: `sudo su -`

1. [Download](#) the tarball appropriate to your operating system and architecture.

Tip: To download packages from the command line, run `wget --content-disposition <URL>` or `curl -JLO <URL>`, using the URL for the tarball you want to download.

2. Unpack the installation tarball:

```
tar -xf <TARBALL_FILENAME>
```

3. From the installer directory, run the installer and follow the CLI instructions to complete your installation:

```
./puppet-enterprise-installer
```

4. Optional: Restart the shell in order to use client tool commands.

Verify the installation package

If your organization requires that you verify authenticity before installing packages, follow these steps to verify the PE installation tarball using GPG.

Before you begin

You must have GnuPG installed to sign for the release key. GnuPG is an open source program that allows you to safely encrypt and sign digital communications. Visit the [GnuPG website](#) to download, or install using your package management system, for example `yum install gnupg`.

1. Import the Puppet public key.

```
uri='https://downloads.puppetlabs.com/puppet-gpg-signing-key.pub'
curl "$uri" | gpg --import
```

See [Usage notes for curl examples](#) for information about forming curl commands.

2. Print the fingerprint of the key.

```
gpg --fingerprint 0x7F438280EF8D349F
```

The primary key fingerprint displays: 6F6B 1550 9CF8 E59E 6E46 9F32 7F43 8280 EF8D 349F.

3. Verify the release signature of the installation package.

```
$ gpg --verify puppet-enterprise-<version>-<platform>.tar.gz.asc
```

The result is similar to:

```
gpg: Signature made Tue 18 Sep 2016 10:05:25 AM PDT using RSA key ID
EF8D349F
gpg: Good signature from "Puppet, Inc. Release Key (Puppet, Inc. Release
Key)"
```

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

Configuration parameters and the `pe.conf` file

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

Tip: You can install PE using a customized `pe.conf` file by running `./puppet-enterprise-installer -c <PATH_TO_pe.conf>`.

The following are examples of valid parameter and value expressions:

Type	Value
FQDNs	"puppet_enterprise::puppet_master_host": "primary.example.com"

Type	Value
Strings	"console_admin_password": "mypassword"
Arrays	["puppet", "puppetlb-01.example.com"]
Booleans	"puppet_enterprise::profile::orchestrator::run_se true Valid Boolean values are true or false (case sensitive, no quotation marks). Note: Don't use Yes (y), No (n), 1, or 0.
JSON hashes	"puppet_enterprise::profile::orchestrator::java_a { "Xmx": "256m", "Xms": "256m" }
Integer	"puppet_enterprise::profile::console::rbac_sessio "60"

Important: Don't use single quotes on parameter values. Use double quotes as shown in the examples.

Installation parameters

These parameters are required for installation.

Tip: To simplify installation, you can keep the default value of `%{::trusted.certname}` for your primary server and provide a console administrator password after running the installer.

puppet_enterprise::puppet_master_host

The FQDN of the node hosting the primary server, for example `primary.example.com`.

Default: `%{::trusted.certname}`

Database configuration parameters

These are the default parameters and values supplied for the PE databases.

This list is intended for reference only; don't change or customize these parameters.

puppet_enterprise::activity_database_name

Name for the activity database.

Default: `pe-activity`

puppet_enterprise::activity_database_read_user

Activity database user that can perform only read functions.

Default: `pe-activity-read`

puppet_enterprise::activity_database_write_user

Activity database user that can perform only read and write functions.

Default: `pe-activity-write`

puppet_enterprise::activity_database_superuser

Activity database superuser.

Default: `pe-activity`

puppet_enterprise::activity_service_migration_db_user

Activity service database user used for migrations.

Default: `pe-activity`

puppet_enterprise::activity_service_regular_db_user

Activity service database user used for normal operations.

Default: pe-activity-write

puppet_enterprise::classifier_database_name

Name for the classifier database.

Default: pe-classifier

puppet_enterprise::classifier_database_read_user

Classifier database user that can perform only read functions.

Default: pe-classifier-read

puppet_enterprise::classifier_database_write_user

Classifier database user that can perform only read and write functions.

pe-classifier-write

puppet_enterprise::classifier_database_super_user

Classifier database superuser.

pe-classifier

puppet_enterprise::classifier_service_migration_db_user

Classifier service user used for migrations.

Default: pe-classifier

puppet_enterprise::classifier_service_regular_db_user

Classifier service user used for normal operations.

Default: pe-classifier-write

puppet_enterprise::orchestrator_database_name

Name for the orchestrator database.

Default: pe-orchestrator

puppet_enterprise::orchestrator_database_read_user

Orchestrator database user that can perform only read functions.

Default: pe-orchestrator-read

puppet_enterprise::orchestrator_database_write_user

Orchestrator database user that can perform only read and write functions.

Default: pe-orchestrator-write

puppet_enterprise::orchestrator_database_super_user

Orchestrator database superuser.

Default: pe-orchestrator

puppet_enterprise::orchestrator_service_migration_db_user

Orchestrator service user used for migrations.

Default: pe-orchestrator

puppet_enterprise::orchestrator_service_regular_db_user

Orchestrator service user used for normal operations.

Default: pe-orchestrator-write

puppet_enterprise::puppetdb_database_name

Name for the PuppetDB database.

Default: pe-puppetdb

puppet_enterprise::rbac_database_name

Name for the RBAC database.

Default: pe-rbac

puppet_enterprise::rbac_database_read_user

RBAC database user that can perform only read functions.

Default: pe-rbac-read

puppet_enterprise::rbac_database_write_user

RBAC database user that can perform only read and write functions.

Default: pe-rbac-write

puppet_enterprise::rbac_database_super_user

RBAC database superuser.

Default: pe-rbac

puppet_enterprise::rbac_service_migration_db_user

RBAC service user used for migrations.

pe-rbac

puppet_enterprise::rbac_service_regular_db_user

RBAC service user used for normal operations.

Default: pe-rbac-write

External PostgreSQL parameters

These parameters are required to install an external PostgreSQL instance. Password parameters can be added to standard installations if needed.

puppet_enterprise::database_host

Agent certname of the node hosting the database component. Don't use an alt name for this value.

puppet_enterprise::database_port

The port that the database is running on.

Default: 5432

puppet_enterprise::database_ssl

true or false. For unmanaged PostgreSQL installations don't use SSL security, set this parameter to false.

Default: true

puppet_enterprise::database_cert_auth

true or false.

Important: For unmanaged PostgreSQL installations don't use SSL security, set this parameter to false.

Default: true

puppet_enterprise::puppetdb_database_password

Password for the PuppetDB database user. Must be a string, such as "mypassword".

puppet_enterprise::classifier_database_password

Password for the classifier database user. Must be a string, such as "mypassword".

puppet_enterprise::classifier_service_regular_db_user

Database user the classifier service uses for normal operations.

Default: pe-classifier

puppet_enterprise::classifier_service_migration_db_user

Database user the classifier service uses for migrations.

Default: `pe-classifier`

puppet_enterprise::activity_database_password

Password for the activity database user. Must be a string, such as `"mypassword"`.

puppet_enterprise::activity_service_regular_db_user

Database user the activity service uses for normal operations.

Default: `"pe-activity"`

puppet_enterprise::activity_service_migration_db_user

Database user the activity service uses for migrations.

Default: `pe-activity`

puppet_enterprise::rbac_database_password

Password for the RBAC database user. Must be a string, such as `"mypassword"`.

puppet_enterprise::rbac_service_regular_db_user

Database user the RBAC service uses for normal operations.

Default: `"pe-rbac"`

puppet_enterprise::rbac_service_migration_db_user

Database user the RBAC service uses for migrations.

Default: `"pe-rbac"`

puppet_enterprise::orchestrator_database_password

Password for the orchestrator database user. Must be a string, such as `"mypassword"`.

puppet_enterprise::orchestrator_service_regular_db_user

Database user the orchestrator service uses for normal operations.

Default: `pe-orchestrator`

puppet_enterprise::orchestrator_service_migration_db_user

Database user the orchestrator service uses for migrations.

Default: `"pe-orchestrator"`

Primary server parameters

Use these parameters to configure and tune the primary server.

pe_install::puppet_master_dnsaltnames

An array of strings that represent the DNS altnames to be added to the SSL certificate generated for the primary server.

Default: `["puppet"]`

pe_install::install::classification::pe_node_group_environment

String indicating the environment that infrastructure nodes are running in. Specify this parameter if you moved your primary server and other infrastructure nodes from the default `production` environment after install. With non-default environments, this setting ensures that your configuration settings are backed up.

Default: `["production"]`

puppet_enterprise::master::recover_configuration::pe_environment

String indicating the environment that infrastructure nodes are running in. Specify this parameter if you moved your primary server and other infrastructure nodes from the default `production` environment after install. With non-default environments, this setting ensures that your configuration settings are backed up.

Default: ["production"]

puppet_enterprise::profile::certificate_authority

Array of additional certificates to be allowed access to the /certificate_statusAPI endpoint. This list is added to the base certificate list.

puppet_enterprise::profile::master::check_for_updates

true to check for updates whenever the pe-puppetserver service restarts, or false.

Default: true

puppet_enterprise::profile::master::code_manager_auto_configure

true to automatically configure the Code Manager service, or false.

puppet_enterprise::profile::master::r10k_remote

String that represents the Git URL to be passed to the r10k.yaml file, for example "git@your.git.server.com:puppet/control.git". The URL can be any URL that's supported by r10k and Git. This parameter is required only if you want r10k configured when PE is installed; it must be specified in conjunction with puppet_enterprise::profile::master::r10k_private_key.

puppet_enterprise::profile::master::r10k_private_key

String that represents the local file system path on the primary server where the SSH private key can be found and used by r10k, for example "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa". This parameter is required only if you want r10k configured when PE is installed; it must be specified in conjunction with puppet_enterprise::profile::master::r10k_remote.

Console and console-services parameters

Use these parameters to customize the behavior of the console and console-services. Parameters that begin with puppet_enterprise::profile can be modified from the console itself. See the configuration methods documents for more information on how to change parameters in the console or Hiera.

puppet_enterprise::profile::console::classifier_synchronization_period

Integer representing, in seconds, the classifier synchronization period, which controls how long it takes the node classifier to retrieve classes from the primary server.

Default: "600" (seconds).

puppet_enterprise::profile::console::rbac_failed_attempts_lockout

Integer specifying how many failed login attempts are allowed on an account before that account is revoked.

Default: "10" (attempts).

puppet_enterprise::profile::console::rbac_password_reset_expiration

Integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user so that they can reset their password.

Default: "24" (hours).

puppet_enterprise::profile::console::rbac_session_timeout

Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console.

Default: "60" (minutes).

puppet_enterprise::profile::console::session_maximum_lifetime

Integer representing the maximum allowable period that a console session can be valid. To not expire before the maximum token lifetime, set to '0'.

Supported units are "s" (seconds), "m" (minutes), "h" (hours), "d" (days), "y" (years). Units are specified as a single letter following an integer, for example "1d"(1 day). If no units are specified, the integer is treated as seconds.

puppet_enterprise::profile::console::console_ssl_listen_port

Integer representing the port that the console is available on.

Default: [443]

puppet_enterprise::profile::console::ssl_listen_address

Nginx listen address for the console.

Default: "0.0.0.0"

puppet_enterprise::profile::console::classifier_prune_threshold

Integer representing the number of days to wait before pruning the size of the classifier database. If you set the value to "0", the node classifier service is never pruned.

puppet_enterprise::profile::console::classifier_node_check_in_storage

"true" to store an explanation of how nodes match each group they're classified into, or "false".

Default: "false"

puppet_enterprise::profile::console::display_local_time

"true" to display timestamps in local time, with hover text showing UTC time, or "false" to show timestamps in UTC time.

Default: "false"

Modify these configuration parameters in `Hiera` or `pe.conf`, not the console:

puppet_enterprise::api_port

SSL port that the node classifier is served on.

Default: [4433]

puppet_enterprise::console_services::no_longer_reporting_cutoff

Length of time, in seconds, before a node is considered unresponsive.

Default: 3600 (seconds)

console_admin_password

The password to log into the console, for example "myconsolepassword".

Default: Specified during installation.

Orchestrator and orchestration services parameters

Use these parameters to configure and tune the orchestrator and orchestration services.

puppet_enterprise::profile::agent::pxp_enabled

true to enable the Puppet Execution Protocol service, which is required to use the orchestrator and run Puppet from the console, or false.

Default: true

puppet_enterprise::profile::bolt_server::concurrency

An integer that determines the maximum number of concurrent requests orchestrator can make to bolt-server.



CAUTION: Do not set a concurrency limit that is higher than the bolt-server limit. This can cause timeouts that lead to failed task runs.

Default: The default value is set to the current value stored for bolt-server.

puppet_enterprise::profile::orchestrator::global_concurrent_compiles

Integer representing how many concurrent compile requests can be outstanding to the primary server, across all orchestrator jobs.

Default: "8" requests

puppet_enterprise::profile::orchestrator::job_prune_threshold

Integer representing the days after which job reports should be removed.

Default: "30" days

puppet_enterprise::profile::orchestrator::pcp_timeout

Integer representing the length of time, in seconds, before timeout when agents attempt to connect to the Puppet Communications Protocol broker in a Puppet run triggered by the orchestrator.

Default: "30" seconds

puppet_enterprise::profile::orchestrator::run_service

true to enable orchestration services, or false.

Default: true

puppet_enterprise::profile::orchestrator::task_concurrency

Integer representing the number of tasks that can run at the same time.

Default: "250" tasks

puppet_enterprise::profile::orchestrator::use_application_services

true to enable application management, or false.

Default: false

puppet_enterprise::pxp_agent::ping_interval

Integer representing the interval, in seconds, between agents' attempts to ping Puppet Communications Protocol brokers.

Default: "120" seconds

puppet_enterprise::pxp_agent::pxp_logfile

String representing the path to the Puppet Execution Protocol agent log file. Change as needed.

Default: /var/log/puppetlabs/pxp-agent/pxp-agent.log (*nix) or C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log (Windows)

Related information

[Configuring Puppet orchestrator](#) on page 469

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

PuppetDB parameters

Use these parameters to configure and tune PuppetDB.

puppet_enterprise::puppetdb::command_processing_threads

Integer representing how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.

Default: Half the number of cores in your system, for example "8".

puppet_enterprise::profile::master::puppetdb_report_processor_ensure

present to generate agent run reports and submit them to PuppetDB, or absent

Default: present

puppet_enterprise::puppetdb_port

Integer in brackets representing the SSL port that PuppetDB listens on.

Default: "[8081]"

puppet_enterprise::profile::puppetdb::node_purge_ttl

"Time-to-live" value before deactivated or expired nodes are deleted, along with all facts, catalogs, and reports for the node. For example, a value of "14d" sets the time-to-live to 14 days.

Default: "14d"

Java parameters

Use these parameters to configure and tune Java.

`puppet_enterprise::profile::master::java_args`

JVM (Java Virtual Machine) memory, specified as a JSON hash, that is allocated to the Puppet Server service, for example { "Xmx": "4096m", "Xms": "4096m" }.

`puppet_enterprise::profile::puppetdb::java_args`

JVM memory, specified as a JSON hash, that is allocated to the PuppetDB service, for example { "Xmx": "512m", "Xms": "512m" }.

`puppet_enterprise::profile::console::java_args`

JVM memory, specified as a JSON hash, that is allocated to console services, for example { "Xmx": "512m", "Xms": "512m" }.

`puppet_enterprise::profile::orchestrator::java_args`

JVM memory, set as a JSON hash, that is allocated to orchestration services, for example, { "Xmx": "256m", "Xms": "256m" }.

Purchasing and installing a license key

Your license must support the number of nodes that you want to manage with Puppet Enterprise.

Complimentary license

You can manage up to 10 nodes at no charge, and no license key is needed. When you have 11 or more active nodes and no license key, license warnings appear in the console until you install an appropriate license key.

Purchased license

To manage 11 or more active nodes, you must purchase a license. After you purchase a license and install a license key file, your licensed node count and subscription expiration date appear on the **License** page.

Note: To support spikes in node usage, four days per calendar month you can exceed your licensed node count up to double the number of nodes you purchased. This increased number of nodes is called your *bursting limit*. You must buy more nodes for your license if you exceed the licensed node count within the bursting limit on more than four days per calendar month, or if you exceed your bursting limit at all. In these cases, license warnings appear in the console until you contact your Puppet representative.

Related information

[How nodes are counted](#) on page 315

Your *node count* is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your *bursting limit*. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

Getting a license

[Contact our sales team](#) to purchase a new license, or to upgrade, renew, or add nodes.

Tip: To reduce your active node count and free up licenses, remove inactive nodes from your deployment. By default, nodes with Puppet agents are automatically deactivated after seven days with no activity (no new facts, catalog, or reports).

Related information

[Remove agent nodes](#) on page 311

If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Install a license key

Install the `license.key` file to upgrade from a test installation to an active installation.

1. Install the license key by copying the file to `/etc/puppetlabs/license.key` on the primary server node.
2. Verify that Puppet has permission to read the license key by checking its ownership and permissions: `ls -la /etc/puppetlabs/license.key`
3. If the ownership is not `root` and permissions are not `-rw-r--r--` (octal 644), set them:

```
sudo chown root:root /etc/puppetlabs/license.key
sudo chmod 644 /etc/puppetlabs/license.key
```

View license details for your environment

Check the number of active nodes in your deployment, the number of licensed nodes you purchased, and the expiration date for your license.

Procedure

- In the console, click **License**.

The **License** page opens with information on your licensed nodes, bursting limit, and subscription expiration date. Any license warnings that appear in the console navigation are explained here. For example, if your license is expired or out of compliance.

Related information

[Puppet orchestrator API: usage endpoint](#) on page 607

Use the `/usage` endpoint to view details about the active nodes in your deployment.

[Remove agent nodes](#) on page 311

If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Installing agents

You can install Puppet Enterprise agents on `*nix`, Windows, and macOS.

To install FIPS-enabled PE agents, install the appropriate FIPS-enabled agent on a third party [supported platform](#) with FIPS mode enabled. You can use FIPS-enabled agents with a non-FIPS enabled primary server.

Agents are typically installed from a package repository hosted on your primary server. The PE package management repository is created during installation of the primary server and serves packages over HTTPS using the same port as the primary server (8140). This means agents don't require any new ports to be open other than the one they already need to communicate with the primary server.

Agent packages can be found on the primary server in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/`. This directory contains the platform specific repository file structure for agent packages. For example, if your primary server is running on CentOS 7, in `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/`, there's a directory `el-7-x86_64`, which contains the directories with all the packages needed to install an agent.

After installing agents, you must sign their certificates.

Using the install script

The install script installs and configures the agent on target nodes using installation packages from the PE package management repo.

The agent install script performs these actions:

- Detects the operating system on which it's running, sets up an apt, yum, or zipper repo that refers back to the primary server, and then pulls down and installs the puppet-agent packages. If the install script can't find agent packages corresponding to the agent's platform, it fails with an error telling you which pe_repo class you need to add to the primary server.
- Downloads a tarball of plugins from the primary server. This feature is controlled by the settings pe_repo::enable_bulk_pluginsync and pe_repo::enable_windows_bulk_pluginsync, which is set to true (enabled) by default. Depending on how many modules you have installed, bulk plug-in sync can speed agent installation significantly.

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

- Creates a basic puppet.conf file.
- Kicks off a Puppet run.

Automatic downloading of agent installer packages and plugins using the pe_repo class requires an internet connection.

Tip: If your primary server uses a proxy server to access the internet, prior to installation, specify pe_repo::http_proxy_host and pe_repo::http_proxy_port in pe.conf, Hiera, or in the console, in the pe_repo class of the **PE Master** node group.

You can customize agent installation by providing as flags to the script any number of these options, in any order:

Option	Example	Result
puppet.conf settings	<pre>agent:splay=true agent:certname=node1.corp.net agent:environment=development</pre>	<p>The puppet.conf file looks like this:</p> <pre>[agent] certname = node1.corp.net splay = true environment = development</pre>
CSR attribute settings	<pre>extension_requests:pp_role=webserver custom_attributes:challengePassword=abc123</pre>	<p>The installer creates a csr_attributes.yaml file before installing with this content:</p> <pre>--- custom_attributes: challengePassword: abc123 extension_requests: pp_role: webserver</pre>
MSI properties (Windows only)	<pre>-PuppetAgentAccountUser 'pup_adm' -PuppetAgentAccountPassword 'secret'</pre>	<p>The Puppet service runs as pup_adm with a password of secret.</p>

Option	Example	Result
Puppet service status	<p>*nix:</p> <pre>--puppet-service-ensure stopped --puppet-service-enable false</pre> <p>Windows</p> <pre>-PuppetServiceEnsure stopped -PuppetServiceEnable false</pre>	The Puppet service is stopped and doesn't boot after installation. An initial Puppet run doesn't occur after installation.

puppet.conf settings

You can specify any agent configuration option using the install script. Configuration settings are added to `puppet.conf`.

These are the most commonly specified agent config options:

- server
- certname
- environment
- splay
- splaylimit
- noop

Tip: On Enterprise Linux systems, if you have a proxy between the agent and the primary server, you can specify `http_proxy_host`, for example `-s agent:http_proxy_host=<PROXY_FQDN>`.

See the [Configuration Reference](#) for details.

CSR attribute settings

These settings are added to `puppet.conf` and included in the `custom_attributes` and `extension_requests` sections of `csr_attributes.yaml`.

You can pass as many parameters as needed. Follow the `section:key=value` pattern and leave one space between parameters.

See the [csr_attributes.yaml](#) reference for details.

*nix install script with example agent setup and certificate signing parameters:

```
uri='https://primary.example.com:8140/packages/current/install.bash'

curl --insecure "$uri" | sudo bash -s agent:certname=<CERTNAME OTHER THAN
FQDN> custom_attributes:challengePassword=<PASSWORD_FOR_AUTOSIGNER_SCRIPT>
extension_requests:pp_role=<PUPPET NODE ROLE>
```

Note: If you are unable to run the installation script, or install the MSI package with `msiexec`, you can set CSR attributes by creating a `csr_attributes.yaml` file in the Puppet confdir (default `C:\ProgramData\PuppetLabs\puppet\etc\csr_attributes.yaml`) prior to installing the Puppet agent package.

See [Usage notes for curl examples](#) for information about forming curl commands.

Windows install script with example agent setup and certificate signing parameters:

```
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_FQDN>:8140/
packages/current/install.ps1', 'install.ps1'); .
\install.ps1 agent:certname=<certnameOtherThanFQDN>
custom_attributes:challengePassword=<passwordForAutosignerScript>
extension_requests:pp_role=<puppetNodeRole>
```

MSI properties (Windows)

For Windows, you can set these MSI properties, with or without additional agent configuration settings.

MSI Property	PowerShell flag
INSTALLDIR	-InstallDir
PUPPET_AGENT_ACCOUNT_USER	-PuppetAgentAccountUser
PUPPET_AGENT_ACCOUNT_PASSWORD	-PuppetAgentAccountPassword
PUPPET_AGENT_ACCOUNT_DOMAIN	-PuppetAgentAccountDomain

Windows install script with MSI properties and agent configuration settings:

```
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_HOSTNAME>:8140/packages/current/
install.ps1', 'install.ps1'); .\install.ps1 -PuppetAgentAccountUser
"svcPuppet" -PuppetAgentAccountPassword "s3kr3t_P@ssword" agent:splay=true
agent:environment=development
```

Puppet service status

By default, the install script starts the Puppet agent service and kicks off a Puppet run. If you want to manually trigger a Puppet run, or you're using a provisioning system that requires non-default behavior, you can control whether the service is running and enabled.

Option	*nix	Windows	Values
ensure	--puppet-service-ensure <VALUE>	- PuppetServiceEnsure <VALUE>	<ul style="list-style-type: none"> • running • stopped
enable	--puppet-service-enable <VALUE>	- PuppetServiceEnable <VALUE>	<ul style="list-style-type: none"> • true • false • manual (Windows only) • mask

For example:

*nix

```
uri='https://<PRIMARY_FQDN>:8140/packages/current/install.bash'
curl --insecure "$uri" | sudo bash -s -- --puppet-service-ensure stopped
```

Windows

```
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_FQDN>:8140/packages/current/
install.ps1', 'install.ps1'); .\install.ps1 -PuppetServiceEnsure stopped
```

Install *nix agents

You have multiple options for installing *nix agents: from the console, from the command line using PE package management or your own package management, with or without internet access, and more. Chose the installation method that best suits your needs.

Note: You must enable TLSv1 to install agents on these platforms:

- AIX
- Solaris 11

Related information

[Enable TLSv1](#) on page 686

TLSv1 and TLSv1.1 are disabled by default in PE.

Install agents from the console

You can use the console to leverage tasks that install *nix or Windows agents on target nodes.

Before you begin

Add classes to the **PE Master** node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.

Make sure you have permission to run the appropriate task to install agents on all nodes:

- *nix targets use the task `pe_bootstrap::linux`
- Windows targets use the task `pe_bootstrap::windows`

For Windows targets, this task requires:

- Windows 2008 SP2 or newer
- PowerShell version 3 or higher
- Microsoft .NET Framework 4.5 or higher

Note: To add additional parameters to the [pe_bootstrap](#) task during the agent installation, click **Advanced install** on the **Install agent on nodes** page.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Install agents**.
3. Select a transport method.
 - **SSH** for *nix targets
 - **WinRM** for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click **Add nodes**.

Tip: Click **Installation job started** to view the job details for the task.

Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

Related information

[Managing certificate signing requests in the console](#) on page 60

A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

[Managing certificate signing requests on the command line](#) on page 131

You can view, approve, and reject node requests using the command line.

Install *nix agents with PE package management

PE provides its own package management to help you install agents in your infrastructure.

Note: The <PRIMARY_HOSTNAME> portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. If you're installing an agent with a different OS than the primary server, add the appropriate class for the repo that contains the agent packages.

- a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
- b) On the **Classes** tab, enter `pe_repo` and select the repo class from the list of classes.

Note: The repo classes are listed as

```
pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>.
```

- c) Click **Add class**, and commit changes.
- d) Run `puppet agent -t` to configure the primary server using the newly assigned class.

The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>/`.

2. SSH into the node where you want to install the agent, and run the installation command appropriate to your environment.

- Curl

```
uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
curl --insecure "$uri" | sudo bash
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Tip: On AIX versions 7.10 and earlier, which don't support the `-k` option, use `--tlsv1` instead. If neither `-k` or `--tlsv1` is supported, you must install using a manually transferred certificate.

- RHEL 5 and CentOS 5

```
uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
curl --insecure --tlsv1 "$uri" | sudo bash
```

- wget

```
wget -O - -q --no-check-certificate https://<PRIMARY_HOSTNAME>:8140/
packages/current/install.bash | sudo bash
```

- Solaris 10 (run as root)

```
export PATH=$PATH:/opt/sfw/bin
wget -O - -q --no-check-certificate --secure-protocol=TLSv1 https://
<PRIMARY_HOSTNAME>:8140/packages/current/install.bash | bash
```

Install *nix agents with your own package management

If you choose not to use PE package management to install agents, you can use your own package management tools.

Before you begin

[Download](#) the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Configure the package manager on your agent node (Yum, Apt) to point to that repo.
3. Install the agent using the command appropriate to your environment.

- Yum

```
sudo yum install puppet-agent
```

- Apt

```
sudo apt-get install puppet-agent
```

Install *nix agents using a manually transferred certificate

If you choose not to or can't use `curl --insecure` to trust the primary server during agent installation, you can manually transfer the primary server CA certificate to any machines you want to install agents on, and then run the installation script against that cert.

1. On the machine that you're installing the agent on, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and transfer `ca.pem` to the `certs` directory you created on the agent node.
3. On the agent node, verify file permissions: `chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem`
4. Run the installation command, using the `--cacert` flag to point to the cert:

```
cacert='/etc/puppetlabs/puppet/ssl/certs/ca.pem'
uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'

curl --cacert "$cacert" "$uri" | sudo bash
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Install *nix agents without internet access

If you don't have access to the internet beyond your infrastructure, you can download the appropriate agent tarball from an internet-connected system and then install using the package management solution of your choice.

Install *nix agents with PE package management without internet access

Use PE package management to install agents when you don't have internet access beyond your infrastructure.

Before you begin

[Download](#) the appropriate agent tarball.

1. On your primary server, copy the agent tarball to `/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-<AGENT_VERSION>`, for example `opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-5.5.17/`
2. If you're installing an agent with a different OS than the primary server, add the appropriate class for the repo that contains the agent packages.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - b) On the **Classes** tab, enter `pe_repo` and select the repo class from the list of classes.

Note: The repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.
 - c) Click **Add class**, and commit changes.
3. Run Puppet: `puppet agent -t`
4. Follow the steps for [Install *nix agents with PE package management](#) on page 117.

Install *nix agents with your own package management without internet access

Use your own package management to install agents when you don't have internet access beyond your infrastructure.

Before you begin

[Download](#) the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Disable the PE-hosted repo.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - b) On the **Classes** tab, find `pe_repo` class (as well as any class that begins `pe_repo::`), and click **Remove this class**.
 - c) Commit changes.

Install *nix agents from compilers using your own package management without internet access

If your infrastructure relies on compilers to install agents, you don't have to copy the agent package to each compiler. Instead, use the console to specify a path to the agent package on your package management server.

Before you begin

[Download](#) the appropriate agent tarball.

1. Add the agent package to your own package management and distribution system.
2. Set the `base_path` parameter of the `pe_repo` class to your package management server.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - b) On the **Classes** tab, find the `pe_repo` class and specify the parameter.

Parameter	Value
<code>base_path</code>	FQDN of your package management server.

- c) Click **Add parameter** and commit changes.

Install Windows agents

You have multiple options for installing Windows agents: from the console, from a PowerShell window using PE package management, from the command line using an .msi package, and more. Chose the installation method that best suits your needs.

Related information

[Enable TLSv1](#) on page 686

TLSv1 and TLSv1.1 are disabled by default in PE.

Install agents from the console

You can use the console to leverage tasks that install *nix or Windows agents on target nodes.

Before you begin

Add classes to the **PE Master** node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.

Make sure you have permission to run the appropriate task to install agents on all nodes:

- *nix targets use the task `pe_bootstrap::linux`
- Windows targets use the task `pe_bootstrap::windows`

For Windows targets, this task requires:

- Windows 2008 SP2 or newer
- PowerShell version 3 or higher

- Microsoft .NET Framework 4.5 or higher

Note: To add additional parameters to the [pe_bootstrap task](#) during the agent installation, click **Advanced install** on the **Install agent on nodes** page.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Install agents**.
3. Select a transport method.
 - **SSH** for *nix targets
 - **WinRM** for Windows targets
4. Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Click **Add nodes**.

Tip: Click **Installation job started** to view the job details for the task.

Agents are installed on the target nodes and then automatically submit certificate signing requests (CSR) to the primary server. The list of unsigned certificates is updated with new targets.

Related information

[Managing certificate signing requests in the console](#) on page 60

A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

[Managing certificate signing requests on the command line](#) on page 131

You can view, approve, and reject node requests using the command line.

Install Windows agents with PE package management

To install a Windows agent with PE package management, you use the `pe_repo` class to distribute an installation package to agents. You can use this method with or without internet access.

Before you begin

If your primary server doesn't have internet access, [download](#) the appropriate agent package and save it on your primary server in the location appropriate for your agent systems:

- 32-bit systems — `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/`
- 64-bit systems — `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/`

You must use PowerShell 2.0 or later to install Windows agents with PE package management.

Note: The `<PRIMARY_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
2. On the **Classes** tab, type `pe_repo` and select the appropriate repo class from the list of classes.
 - 64-bit (x86_64) — `pe_repo::platform::windows_x86_64`.
 - 32-bit (i386) — `pe_repo::platform::windows_i386`.
3. Click **Add class** and commit changes.
4. On the primary server, run Puppet to configure the newly assigned class.

The new repository is created on the primary server at `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>/`.

- On the node, open an administrative PowerShell window, and install:

```
[System.Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12;
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_HOSTNAME>:8140/packages/
current/install.ps1', 'install.ps1'); .\install.ps1 -v
```

Microsoft Windows Server 2008r2 only:

```
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_HOSTNAME>:8140/packages/
current/install.ps1', 'install.ps1'); .\install.ps1 -v
```

After running the installer, you see the following output, which indicates the agent was successfully installed.

```
Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'
service { 'puppet':
  ensure => 'running',
  enable => 'true',
}
```

Install Windows agents using a manually transferred certificate

If you need to perform a secure installation on Windows nodes, you can manually transfer the primary server CA certificate to target nodes, and run a specialized installation script against that cert.

- Transfer the installation script and the CA certificate from your primary server to the node you're installing.

File	Location on primary server	Location on target node
Installation script (install.ps1)	/opt/puppetlabs/server/data/packages/public/	Any accessible local directory.
CA certificate (ca.pem)	/etc/puppetlabs/puppet/ssl/certs/	C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\

- Run the installation script, using the `-UsePuppetCA` flag: `.\install.ps1 -UsePuppetCA`

Install Windows agents with the .msi package

Use the Windows .msi package if you need to specify agent configuration details during installation, or if you need to install Windows agents locally without internet access.

Before you begin

[Download](#) the .msi package.

Tip: To install on nodes that don't have internet access, save the .msi package to the appropriate location for your system:

- 32-bit systems — /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/
- 64-bit systems — /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/

Install Windows agents with the installer

Use the MSI installer for a more automated installation process. The installer can configure `puppet.conf`, create CSR attributes, and configure the agent to talk to your primary server.

1. Run the installer as administrator.
2. When prompted, provide the hostname of your primary server, for example `puppet`.

Install Windows agents using `msiexec` from the command line

Install the MSI manually from the command line if you need to customize `puppet.conf`, CSR attributes, or certain agent properties.

On the command line of the node you want to install the agent on, run the install command:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi
```

Tip:

- You can specify `/l*v install.txt` to log the progress of the installation to a file.
- You can set CSR attributes by creating a `csr_attributes.yaml` file in the Puppet confdir (default `C:\ProgramData\PuppetLabs\puppet\etc\csr_attributes.yaml`) prior to installing the Puppet agent package.

MSI properties

If you install Windows agents from the command line using the `.msi` package, you can optionally specify these properties.

Important: If you set a non-default value for `PUPPET_MASTER_SERVER`, `PUPPET_CA_SERVER`, `PUPPET_AGENT_CERTNAME`, or `PUPPET_AGENT_ENVIRONMENT`, the installer replaces the existing value in `puppet.conf` and re-uses the value at upgrade unless you specify a new value. Therefore, if you've customized these properties, don't change the setting directly in `puppet.conf`; instead, re-run the installer and set a new value at installation.

Property	Definition	Setting in <code>pe.conf</code>	Default
<code>INSTALLDIR</code>	Location to install Puppet and its dependencies.	n/a	<ul style="list-style-type: none"> • 32-bit — <code>C:\Program Files\Puppet Labs\Puppet</code> • 64-bit — <code>C:\Program Files\Puppet Labs\Puppet</code>
<code>PUPPET_MASTER_SERVER</code>	Hostname where the primary server can be reached.	<code>server</code>	<code>puppet</code>
<code>PUPPET_CA_SERVER</code>	Hostname where the CA primary server can be reached, if you're using multiple primary servers and only one of them is acting as the CA.	<code>ca_server</code>	Value of <code>PUPPET_MASTER_SERVER</code>

Property	Definition	Setting in <code>pe.conf</code>	Default
PUPPET_AGENT_CERTNAME	<p>Node's certificate name, and the name it uses when requesting catalogs.</p> <p>For best compatibility, limit the value of <code>certname</code> to lowercase letters, numbers, periods, underscores, and dashes.</p>	<code>certname</code>	Value of <code>facter fqdn</code>
PUPPET_AGENT_ENVIRONMENT	<p>Node's environment.</p> <p>Note: If a value for the <code>environment</code> variable already exists in <code>puppet.conf</code>, specifying it during installation does not override that value.</p>	<code>environment</code>	<code>production</code>

Property	Definition	Setting in <code>pe.conf</code>	Default
PUPPET_AGENT_STARTUP	<p>Whether and how the agent service is allowed to run. Allowed values are:</p> <ul style="list-style-type: none"> • <code>Automatic</code> — Agent starts up when Windows starts and remains running in the background. • <code>Manual</code> — Agent can be started in the services console or with <code>net start</code> on the command line. • <code>Disabled</code> — Agent is installed but disabled. You must change its startup type in the services console before you can start the service. 	n/a	<code>Automatic</code>

Property	Definition	Setting in <code>pe.conf</code>	Default
PUPPET_AGENT_ACCOUNT_USER	<p>Windows user account the agent service uses. This property is useful if the agent needs to access files on UNC shares, because the default LocalService account can't access these network resources.</p> <p>The user account must already exist, and can be either a local or domain user. The installer allows domain users even if they have not accessed the machine before. The installer grants Logon as Service to the user, and if the user isn't already a local administrator, the installer adds it to the Administrators group.</p> <p>This property must be combined with PUPPET_AGENT_ACCOUNT_PASSWORD and PUPPET_AGENT_ACCOUNT_DOMAIN.</p>	n/a	LocalSystem
PUPPET_AGENT_ACCOUNT_PASSWORD	Password for the agent's user account.	n/a	No Value
PUPPET_AGENT_ACCOUNT_DOMAIN	Domain of the agent's user account.	n/a	.
REINSTALLMODE	<p>A default MSI property used to control the behavior of file copies during installation.</p> <p>Important: If you need to downgrade agents, use REINSTALLMODE=amus when calling <code>msiexec.exe</code> at the command line to prevent removing files that the application needs.</p>	n/a	<p>amus as of puppet-agent 1.10.10 and puppet-agent 5.3.4</p> <p>omus in prior releases</p>

To install the agent with the primary server at `puppet.acme.com`:

```
msiexec /qn /norestart /i puppet.msi PUPPET_MASTER_SERVER=puppet.acme.com
```

To install the agent to a domain user `ExampleCorp\bob`:

```
msiexec /qn /norestart /i puppet-<VERSION>.msi
  PUPPET_AGENT_ACCOUNT_DOMAIN=ExampleCorp PUPPET_AGENT_ACCOUNT_USER=bob
  PUPPET_AGENT_ACCOUNT_PASSWORD=password
```

Windows agent installation details

Windows nodes can fetch configurations from a primary server and apply manifests locally, and respond to orchestration commands.

After installing a Windows node, the **Start Menu** contains a **Puppet** folder with shortcuts for running the agent manually, running `Factor`, and opening a command prompt for use with Puppet tools.

Note: You must run Puppet with elevated privileges. Select **Run as administrator** when opening the command prompt.

The agent runs as a Windows service. By default, the agent fetches and applies configurations every 30 minutes. The agent service can be started and stopped independently using either the service control manager UI or the command line `sc.exe` utility.

Puppet is automatically added to the machine's `PATH` environment variable, so you can open any command line and run `puppet`, `factor` and the other batch files that are in the `bin` directory of the Puppet installation. Items necessary for the Puppet environment are also added to the shell, but only for the duration of execution of each of the particular commands.

The installer includes Ruby, gems, and `Factor`. If you have existing copies of these applications, such as Ruby, they aren't affected by the re-distributed version included with Puppet.

Program directory

Unless overridden during installation, PE and its dependencies are installed in `Program Files` at `\Puppet Labs\Puppet`.

You can locate the `Program Files` directory using the `PROGRAMFILES` variable or the `PROGRAMFILES(X86)` variable.

The program directory contains these subdirectories.

Subdirectory	Contents
<code>bin</code>	scripts for running Puppet and <code>Factor</code>
<code>factor</code>	<code>Factor</code> source
<code>hiera</code>	Hiera source
<code>misc</code>	resources
<code>puppet</code>	Puppet source
<code>service</code>	code to run the agent as a service
<code>sys</code>	Ruby and other tools

Data directory

PE stores settings, manifests, and generated data — such as logs and catalogs — in the data directory. The data directory contains two subdirectories for the various components:

- `etc` (the `$confdir`): Contains configuration files, manifests, certificates, and other important files.

- `var` (the `$vardir`): Contains generated data and logs.

When you run Puppet with elevated privileges as intended, the data directory is located in the `COMMON_APPDATA.aspx` folder. This folder is typically located at `C:\ProgramData\PuppetLabs\`. Because the common app data directory is a system folder, it is hidden by default.

If you run Puppet without elevated privileges, it uses a `.puppet` directory in the current user's home folder as its data directory, which can result in unexpected settings.

Install macOS agents

You can install macOS agents with PE package management, from Finder, or from the command line.

To install macOS agents with PE package management, follow the steps to [Install *nix agents with PE package management](#) on page 117.

Important: For macOS agents, the certname is derived from the name of the machine (such as `My-Example-Mac`). To prevent installation issues, make sure the name of the node uses lowercase letters. If you don't want to change your computer's name, you can enter the agent certname in all lowercase letters when prompted by the installer.

Install macOS agents from Finder

You can use Finder to install the agent on your macOS machine.

Before you begin

[Download](#) the appropriate agent tarball.

1. Open the agent package `.dmg` and click the installer `.pkg`.
2. Follow prompts in the installer dialog.

You must include the primary server hostname and the agent certname.

Install macOS agents from the command line

You can use the command line to install the agent on a macOS machine.

Before you begin

[Download](#) the appropriate agent tarball.

1. SSH into the node as a root or sudo user.
2. Mount the disk image: `sudo hdiutil mount <DMGFILE>`

A line appears ending with `/Volumes/puppet-agent-VERSION`. This directory location is the mount point for the virtual volume created from the disk image.

3. Change to the directory indicated as the mount point in the previous step, for example: `cd /Volumes/puppet-agent-VERSION`
4. Install the agent package: `sudo installer -pkg puppet-agent-installer.pkg -target /`
5. Verify the installation: `/opt/puppetlabs/bin/puppet --version`
6. Configure the agent to connect to the primary server: `/opt/puppetlabs/bin/puppet config set server <PRIMARY_HOSTNAME>`
7. Configure the agent certname: `/opt/puppetlabs/bin/puppet config set certname <AGENT_CERTNAME>`

macOS agent installation details

macOS agents include core Puppet functionality, plus platform-specific capabilities like package installation, service management with LaunchD, facts inventory with the System Profiler, and directory services integration.

Install non-root agents

Running agents without root privileges can assist teams using PE to work autonomously.

For example, your infrastructure's platform might be maintained by one team with root privileges while your infrastructure's applications are managed by a separate team (or teams) with diminished privileges. If the application team wants to be able to manage its part of the infrastructure independently, they can run Puppet without root privileges.

PE is installed with root privileges, so you need a root user to install and configure non-root access to a primary server. The root user who performs this installation can then set up non-root users on the primary server and any nodes running an agent.

Non-root users can perform a reduced set of management tasks, including configuring settings, configuring Facter external facts, running `puppet agent --test`, and running Puppet with non-privileged cron jobs or a similar scheduling service. Non-root users can also classify nodes by writing or editing manifests in the directories where they have write privileges.

Install non-root *nix agents

Note: Unless specified otherwise, perform these steps as a root user.

1. Install the agent on each node that you want to operate as a non-root user.
2. Log in to the agent node and add the non-root user:

```
puppet resource user <UNIQUE NON-ROOT USERNAME> ensure=present
managehome=true
```

Note: Each non-root user must have a unique name.

3. Set the non-root user password.

For example, on most *nix systems: `passwd <USERNAME>`

4. Stop the puppet service:

```
puppet resource service puppet ensure=stopped enable=false
```

By default, the puppet service runs automatically as a root user, so it must be disabled.

5. Disable the Puppet Execution Protocol agent.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent** group.
 - b) On the **Classes** tab, select the **puppet_enterprise::profile::agent** class, specify parameters, click **Add parameter**, and then commit changes.

Parameter	Value
pxp_enabled	false

6. Change to the non-root user and generate a certificate signing request:

```
puppet agent -t --certname "<UNIQUE NON-ROOT USERNAME.HOSTNAME>" --server
"<PRIMARY_HOSTNAME>"
```

Tip: If you wish to use `su - <NON-ROOT USERNAME>` to switch between accounts, make sure to use the `-` (`-l` in some unix variants) argument so that full login privileges are correctly granted. Otherwise you might see permission denied errors when trying to apply a catalog.

7. On the primary server or in the console, approve the certificate signing request.

- On the agent node as the non-root user, set the node's certname and the primary server hostname, and then run Puppet.

```
puppet config set certname <UNIQUE NON-ROOT USERNAME.HOSTNAME> --section agent
```

```
puppet config set server <PRIMARY_HOSTNAME> --section agent
```

```
puppet agent -t
```

The configuration specified in the catalog is applied to the node.

Tip: If you see Facter facts being created in the non-root user's home directory, you have successfully created a functional non-root agent.

To confirm that the non-root agent is correctly configured, verify that:

- The agent can request certificates and apply the catalog from the primary server when a non-root user runs Puppet: `puppet agent -t`
- The agent service is not running: `service puppet status`
- Non-root users can collect existing facts by running `facter` on the agent, and they can define new, external facts.

Install non-root Windows agents

Note: Unless specified otherwise, perform these steps as a root user.

- Install the agent on each node that you want to operate as a non-root user.
- Log in to the agent node and add the non-root user:

```
puppet resource user <UNIQUE NON-ADMIN USERNAME> ensure=present  
managehome=true password="puppet" groups="Users"
```

Note: Each non-root user must have a unique name.

- Stop the puppet service:

```
puppet resource service puppet ensure=stopped enable=false
```

By default, the puppet service runs automatically as a root user, so it must be disabled.

- Change to the non-root user and generate a certificate signing request:

```
puppet agent -t --certname "<UNIQUE NON-ADMIN USERNAME>" --server  
"<PRIMARY_HOSTNAME>"
```

Important: This Puppet run submits a cert request to the primary server and creates a `/.puppet` directory structure in the non-root user's home directory. If this directory is not created automatically, you must manually create it before continuing.

- As the non-root user, create a configuration file at `%USERPROFILE%/.puppet/puppet.conf` to specify the agent certname and the hostname of the primary server:

```
[main]  
certname = <UNIQUE NON-ADMIN USERNAME.hostname>  
server = <PRIMARY_HOSTNAME>
```

- As the non-root user, submit a cert request: `puppet agent -t`.

7. On the primary server or in the console, approve the certificate signing request.

Important: It's possible to sign the root user certificate in order to allow that user to also manage the node.

However, this introduces the possibility of unwanted behavior and security issues. For example, if your `site.pp` has no default node configuration, running the agent as non-admin could lead to unwanted node definitions getting generated using alt hostnames, a potential security issue. If you deploy this scenario, ensure the root and non-root users never try to manage the same resources, have clear-cut node definitions, ensure that classes scope correctly, and so forth.

8. On the agent node as the non-root user, run Puppet: `puppet agent -t`.

The configuration specified in the catalog is applied to the node.

Non-root user functionality

Non-root users can use a subset of functionality. Any operation that requires root privileges, such as installing system packages, can't be managed by a non-root agent.

*nix non-root functionality

On *nix systems, as non-root agent you can enforce these resource types:

- `cron` (only non-root cron jobs can be viewed or set)
- `exec` (cannot run as another user or group)
- `file` (only if the non-root user has read/write privileges)
- `notify`
- `schedule`
- `ssh_key`
- `ssh_authorized_key`
- `service`
- `augeas`

Note: When running a cron job as non-root user, using the `-u` flag to set a user with root privileges causes the job to fail, resulting in this error message:

```
Notice: /Stage[main]/Main/Node[nonrootuser]/Cron[illegal_action]/ensure:
created must be privileged to use -u
```

You can also inspect these resource types (use `puppet resource <resource type>`):

- `host`
- `mount`
- `package`

Windows non-root functionality

On Windows systems as non-admin user, you can enforce these types :

- `exec`
- `file`

Note: A non-root agent on Windows is extremely limited as compared to non-root *nix. While you can use the above resources, you are limited on usage based on what the agent user has access to do (which isn't much). For instance, you can't create a file/directory in `C:\Windows` unless your user has permission to do so.

You can also inspect these resource types (use `puppet resource <resource type>`):

- `host`
- `package`
- `user`
- `group`

- `service`

Managing certificate signing requests

When you install a Puppet agent on a node, the agent automatically submits a certificate signing request (CSR) to the primary server. You must accept this request to bring before the node under PE management can be added your deployment. This allows Puppet to run on the node and enforce your configuration, which in turn adds node information to PuppetDB and makes the node available throughout the console.

You can approve certificate requests from the PE console or the command line. If DNS altnames are set up for agent nodes, you must approve the CSRs on use the command line interface .

Note: Specific user permissions are required to manage certificate requests:

- To accept or reject CSRs in the console or on the command line, you need the permission **Certificate requests: Accept and reject**.
- To manage certificate requests in the console, you also need the permission **Console: View**.

Related information

[Installing agents](#) on page 112

You can install Puppet Enterprise agents on *nix, Windows, and macOS.

Managing certificate signing requests in the console

A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

- To manage requests individually, click **Accept** or **Reject**.
- To manage the entire list of requests, click **Accept All** or **Reject All**. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

Related information

[Running Puppet on demand](#) on page 476

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

Managing certificate signing requests on the command line

You can view, approve, and reject node requests using the command line.

To view pending node requests on the command line:

```
$ sudo puppetserver ca list
```

To sign a pending request:

```
$ sudo puppetserver ca sign --certname <NAME>
```

Note: You can use the Puppet Server CA CLI to sign certificates with altnames or auth extensions by default.

Configuring agents

You can add additional configuration to agents by editing `/etc/puppetlabs/puppet/puppet.conf` directly, or by using the `puppet config set` sub-command, which edits `puppet.conf` automatically.

For example, to point the agent at a primary server called `primary.example.com`, run `puppet config set server primary.example.com`. This command adds the setting `server = primary.example.com` to the `[main]` section of `puppet.conf`.

To set the certname for the agent, run `/opt/puppetlabs/bin/puppet config set certname agent.example.com`.

Installing compilers

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

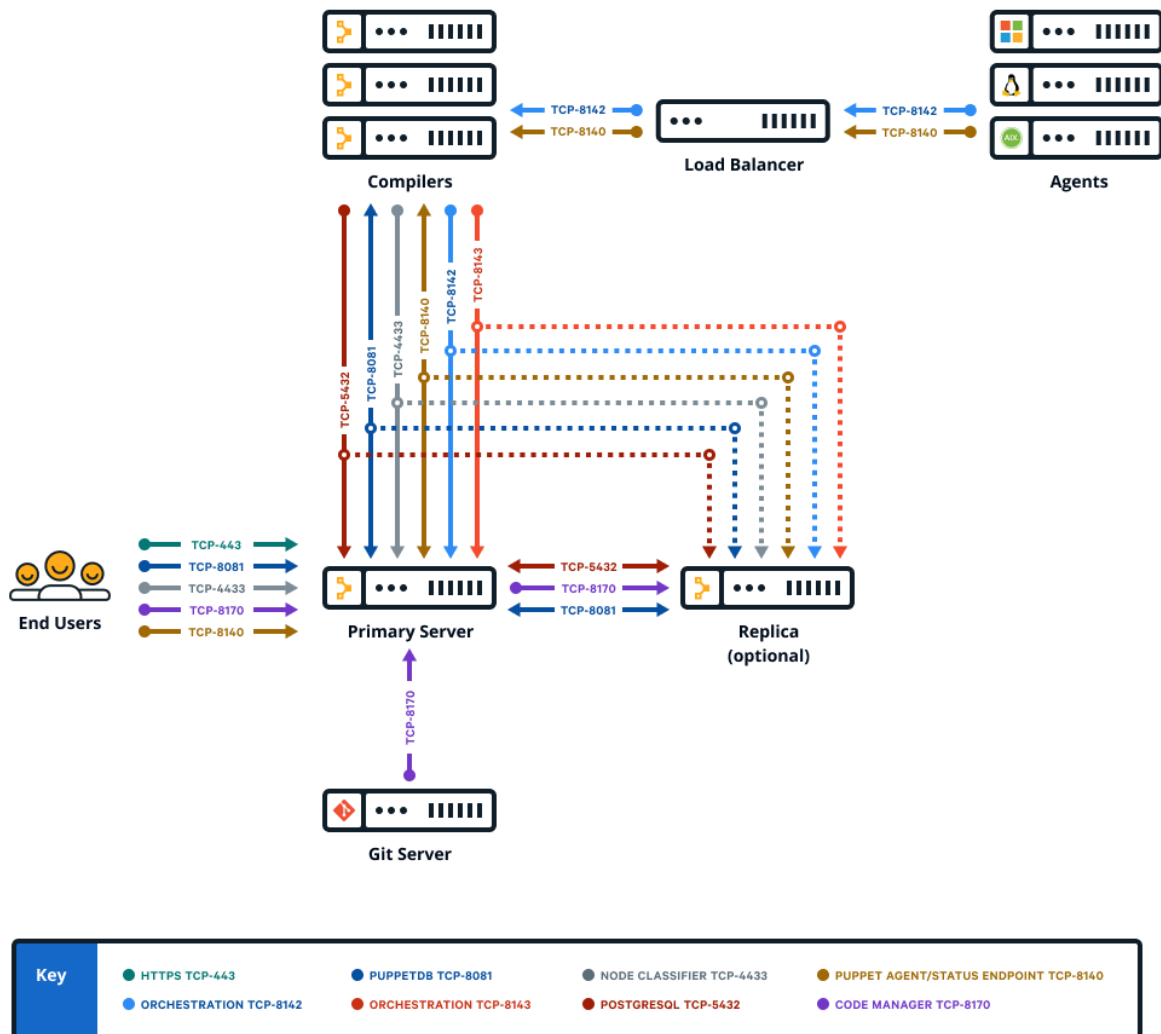
Each compiler increases capacity by 1,500 to 3,000 nodes, until you exhaust the capacity of PuppetDB or the console.

How compilers work

A single primary server can process requests and compile code for up to 4,000 nodes. When you exceed this scale, expand your infrastructure by adding compilers to share the workload and compile catalogs faster.

Important: Compilers must run the same OS major version, platform, and architecture as the primary server.

Compilers act as PCP brokers, conveying messages between the orchestrator and Puppet Execution Protocol (PXP) agents. PXP agents connect to PCP brokers running on compilers over port 8142. Status checks on compilers must be sent to port 8140, using `https://<hostname>:8140/status/v1/simple`.



Components and services running on compilers

Compilers typically run Puppet Server and PuppetDB services, as well as a file sync client. Older, legacy-style compilers must be converted in order to add PuppetDB.

When triggered by a web endpoint, file sync takes changes from the working directory on the primary server and deploys the code to a live code directory. File sync then deploys that code to all your compilers. By default, compilers check for code updates every five seconds.

The certificate authority (CA) service is disabled on compilers. A proxy service running on the compiler Puppet Server directs CA requests to the primary server, which hosts the CA in default installations.

Compilers also have:

- The repository for agent installation, `pe_repo`
- The controller profile used with PE client tools
- Puppet Communications Protocol (PCP) brokers to enable orchestrator scale

Logs for compilers are located at `/var/log/puppetlabs/puppetserver/`.

Logs for PCP brokers on compilers are located at `/var/log/puppetlabs/puppetserver/pcp-broker.log`.

Using load balancers with compilers

When using more than one compiler, a load balancer can help distribute the load between the compilers and provide a level of redundancy.

Specifics on how to configure a load balancer infrastructure falls outside the scope of this document, but examples of how to leverage haproxy for this purpose can be found in the HAproxy module documentation.

Calculating load balancing

PCP brokers run on compilers and connect to PXP agents over port 8142. PCP brokers are built on websockets and require many persistent connections. If you're not using HTTP health checks, we recommend using a round robin or random load balancing algorithm for PXP agent connections to PCP brokers, because PCP brokers don't operate independent of the orchestrator and isolate themselves if they become disconnected. You can check connections with the `/status/v1/simple` endpoint for an error state.

Configure your load balancer to avoid closing long-lived connections that have little traffic. In the HAproxy module, you can set the `timeout tunnel` to 15m because PCP brokers disconnect inactive connections after 15 minutes.

Due to the diverse nature of the network communications between the agent and the primary server, we recommend that you implement a load balancing algorithm that distributes traffic between compilers based on the number of open connections. Load balancers often refer to this strategy as "balancing by least connections."

Using health checks

The Puppet REST API exposes a status endpoint that can be leveraged from a load balancer health check to ensure that unhealthy hosts do not receive agent requests from the load balancer.

The primary server service responds to unauthenticated HTTP GET requests issued to `https://<hostname>:8140/status/v1/simple`. The API responds with an HTTP 200 status code if the service is healthy.

If your load balancer doesn't support HTTP health checks, a simpler alternative is to check that the host is listening for TCP connections on port 8140. This ensures that requests aren't forwarded to an unreachable instance of the primary server, but it does not guarantee that a host is pulled out of rotation if it's deemed unhealthy, or if the service listening on port 8140 is not a service related to Puppet.

Load balancing for geodiverse locations

If you have load balancers in multiple data centers, set the `pe_repo::compile_master_pool_address` in Hieradata at the point of locational demarcation. With multiple data locations, specify this value at the lowest point in your hierarchy that still accurately provides appropriate values for each location. For example, if `continent` is your demarcation, your `common.yaml` Hieradata file might specify:

```
[datadir]/continent/europe.yaml
[datadir]/continent/australia.yaml
```

And your continent-specific `.yaml` files would specify:

```
pe_repo::compile_master_pool_address: <LOAD_BALANCER_CERTNAME>
```

Related information

[Firewall configuration for large installations](#) on page 90

These are the port requirements for large installations with compilers.

[GET /status/v1/simple](#) on page 300

The `/status/v1/simple` returns a status that reflects all services the status service knows about.

[Configure settings with Hiera](#) on page 161

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

Install compilers

Installing a compiler adds the specified node to the **PE Infrastructure Agent** and **PE Compiler** node groups and installs the PuppetDB service on the node.

Before you begin

The node you want to provision as a compiler must have a Puppet agent installed, or you must be able to connect to a non-agent node with SSH.

To install a FIPS-compliant compiler, install the compiler on a [supported platform](#) with FIPS mode enabled. The node must be configured with sufficient available entropy or the installation process fails.

Important: Contact Support for guidance before installing compilers in geo-diverse installations. If your primary server and compilers are connected with high-latency links or congested network segments, you might experience better PuppetDB performance with legacy compilers.

1. Configure the agent on infrastructure nodes to connect to the primary server.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent > PE Infrastructure Agent** group.
 - b) If you manage your load balancers with agents, on the **Rules** tab, pin load balancers to the group. Pinning load balancers to the **PE Infrastructure Agent** group ensures that they communicate directly with the primary server.
 - c) On the **Classes** tab, find the `puppet_enterprise::profile::agent` class and specify these parameters:

Parameter	Value
<code>manage_puppet_conf</code>	Specify <code>true</code> to ensure that your setting for <code>server_list</code> is configured in the expected location and persists through Puppet runs.
<code>pcp_broker_list</code>	Hostname for your primary server and replica, if you have one. Hostnames must include port 8142, for example ["PRIMARY.EXAMPLE.COM:8142" , "REPLICA.EXAMPLE.COM:8142"].
<code>master_uris</code> <code>server_list</code>	Hostname for your primary server and replica, if you have one, for example ["PRIMARY.EXAMPLE.COM" , "REPLICA.EXAMPLE.COM"]. This setting assumes port 8140 unless you specify otherwise with <code>host:port</code> .

- d) Remove any values set for `pcp_broker_ws_uris`.
 - e) Commit changes.
 - f) Run Puppet on all agents classified into the **PE Infrastructure Agent** group.
2. Pin the node that you want to provision to the **PE Infrastructure Agent** group and run Puppet on the node:


```
puppet agent -t
```

3. On your primary server logged in as root, run:

```
puppet infrastructure provision compiler <COMPILER_FQDN>
```

You can specify these optional parameters:

- `dns-alt-names` — Comma-separated list of any alternative names that agents use to connect to compilers. The installation uses `puppet` by default.

Note: If your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns-alt-names` parameter when provisioning your compiler.

- `no-dns-alt-names` — Prevents the installer from setting the default alternative name, `puppet`. Use this parameter if you don't allow alternative names (`allow-subject-alt-names: false` in your `ca.conf`).
- `use-ssh` — Enables installing on a node that doesn't have a Puppet agent currently installed. You must be able to connect to the node with SSH. You can pair this flag with additional SSH parameters. Run `puppet infrastructure provision --help` for details.

Configure compilers to appropriately route communication between your primary server and agent nodes.

Configure compilers

Compilers must be configured to appropriately route communication between your primary server and agent nodes.

Before you begin

- Install compilers and load balancers.
 - If you need DNS altnames for your load balancers, add them to the primary server.
 - Ensure port 8143 is open on the primary server or on any workstations used to run orchestrator jobs.
1. Configure `pe_repo::compile_master_pool_address` to send agent install requests to the load balancer.

Important: If you have load balancers in multiple data centers, you must configure `compile_master_pool_address` using Hiera, instead of using configuration data in the console, as described in this step. Using either of these methods updates the agent install script URL displayed in the console.

Note: If you are using a single compiler, configure `compile_master_pool_address` with the compiler's fully qualified domain name (FQDN).

- a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
- b) On the **Configuration data** tab, select the **pe_repo** class and specify parameters:

Parameter	Value
<code>compile_master_pool_address</code>	Load balancer hostname.

- c) Click **Add data** and commit changes.
2. Run Puppet on the compiler, and then on the primary server.

3. Configure agents to connect orchestration agents to the load balancer.

- a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent** group.
- b) On the **Classes** tab, find the **puppet_enterprise::profile::agent** class and specify parameters:

Parameter	Value
manage_puppet_conf	Specify true to ensure that your setting for <code>server_list</code> is configured in the expected location and persists through Puppet runs.
pcp_broker_list	Hostname for your load balancers. Hostnames must include port 8142, for example ["LOADBALANCER1.EXAMPLE.COM:8142", "LOADBALANCER2.EXAMPLE.COM:8142"]
master_uris	Hostname for your load balancers, for example ["LOADBALANCER1.EXAMPLE.COM", "LOADBALANCER2.EXAMPLE.COM"]
server_list	This setting assumes port 8140 unless you specify otherwise with <code>host:port</code> .

- c) Remove any values set for **pcp_broker_ws_uris**.
- d) Commit changes.
- e) Run Puppet on the primary server, then run Puppet on all agents, or install new agents.

This Puppet run configures PXP agents to connect to the load balancer.

Related information

[Firewall configuration for large installations](#) on page 90

These are the port requirements for large installations with compilers.

[Configure settings with Hiera](#) on page 161

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system.

When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

Convert existing compilers

If you have legacy compilers, you can improve their usability and scalability by adding PuppetDB. In addition to installing the PuppetDB service, converting an existing compiler adds the node to the **PE Compiler** node group and unpins it from the **PE Master** node group.

Before you begin

Open port 5432 from compilers to your primary server or, in extra-large installations, your PE-PostgreSQL node.

Important: Contact Support for guidance before converting compilers in geo-diverse installations. If your primary server and compilers are connected with high-latency links or congested network segments, you might experience better PuppetDB performance with legacy compilers.

On your primary server logged in as root, run: .

```
puppet infrastructure run convert_legacy_compiler
  compiler=<COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

Tip: To convert all compilers:

```
puppet infrastructure run convert_legacy_compiler all=true
```

Run `puppet infrastructure tune` on your primary server and adjust tuning for compilers as needed.

Installing PE client tools

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

The `pe-client-tools` package is included in the PE installation tarball. When you install, the client tools are automatically installed on the same node as the primary server. When you upgrade, client tools are automatically updated on infrastructure nodes and managed nodes, but on *unmanaged* nodes, you must re-install the version of client tools that matches the PE version you upgraded to.

Client tools versions align with PE versions. For example, if you're running PE 2019.8, use the 2019.8 client tools. In some cases, we might issue patch releases ("x.y.z") for PE or the client tools. You don't need to match patch numbers between PE and the client tools. Only the "x.y" numbers need to match.

Note: To see the version of client tools installed on your system, use the command appropriate for your package manager or operating system. For example, on Red Hat: `rpm -q pe-client-tools`.

The package includes client tools for these services:

- **Orchestrator** — Allow you to control the rollout of changes in your infrastructure, and provides the interface to the orchestration service. Tools include `puppet-job` and `puppet-app`.
- **Puppet access** — Authenticates you to the PE RBAC token-based authentication service so that you can use other capabilities and APIs.
- **Code Manager** — Provides the interface for the Code Manager and file sync services. Tools include `puppet-code`.
- **PuppetDB CLI** — Enables certain operations with PuppetDB, such as building queries and handling exports.

Because you can safely run these tools remotely, you no longer need to SSH into the primary server to execute commands. Your permissions to see information and to take action are controlled by PE role-based access control. Your activity is logged under your username rather than under `root` or the `pe-puppet` user.

Related information

[Orchestrator configuration files](#) on page 472

The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate work station, there are two types of configuration files: a global configuration file and a user-specified configuration file.

[Configure puppet-access](#) on page 218

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. If you want to use `puppet-access`, ensure it is configured correctly before using it to generate authentication tokens.

[Installing and configuring puppet-code](#) on page 639

PE automatically installs and configures the `puppet-code` command on your primary server as part of the included PE client tools package. You can also set up `puppet-code` on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.

Supported PE client tools operating systems

The PE client tools package can be installed on these platforms.

Operating system	Versions	Arch
CentOS	6, 7, 8	x86_64
Oracle Linux	6, 7, 8	x86_64
Red Hat Enterprise Linux	6, 7, 8	x86_64
Scientific Linux	6, 7	x86_64
SUSE Linux Enterprise Server	12	x86_64

Operating system	Versions	Arch
Ubuntu	16.04, 18.04	amd64
Microsoft Windows	10	x86, x64
Microsoft Windows Server	2012, 2012 R2, 2012 R2 core	x64
	2016, 2016 core	
macOS	10.14, 10.15	

Install PE client tools on a managed workstation

To use the client tools on a system other than the primary server, where they're installed by default, you can install the tools on a *controller node*.

Before you begin

Controller nodes must be running the same OS as your primary server and must have an agent installed.

1. In the console, create a controller classification group, for example `PE Controller`, and ensure that its **Parent name** is set to **All Nodes**.
2. Select the controller group and add the `puppet_enterprise::profile::controller` class.
3. Pin the node that you want to be a controller to the controller group.
 - a) In the controller group, on the **Rules** tab, in the **Certname** field, enter the certname of the node.
 - b) Click **Pin node** and commit changes.
4. Run Puppet on the controller machine.

Related information

[Create classification node groups](#) on page 319

Create classification node groups to assign classification data to nodes.

Install PE client tools on an unmanaged workstation

You can install the `pe-client-tools` package on any workstation running a supported OS. The workstation OS does not need to match the primary server OS.

Before you begin

Review prerequisites for timekeeping, name resolution, and firewall configuration, and ensure that these ports are available on the workstation.

- **8143** — The orchestrator client uses this port to communicate with orchestration services running on the primary server.
- **4433** — The Puppet access client uses this port to communicate with the RBAC service running on the primary server.
- **8170** — If you use the Code Manager service, it requires this port.

Install PE client tools on an unmanaged Linux workstation

1. On the workstation, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the directory you created on the workstation.
3. On the workstation, make sure file permissions are correct: `chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem`
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary server.
5. Download the [pe-client-tools package](#) for the platform appropriate to your workstation.

6. Use your workstation's package management tools to install the `pe-client-tools`.

For example, on RHEL platforms: `rpm -Uvh pe-client-tools-<VERSION-and-PLATFORM>.rpm`

Install PE client tools on an unmanaged Windows workstation

You can install the client tools on a Windows workstation using the setup wizard or the command line.

To start using the client tools on your Windows workstation, open the **PE ClientTools Console** from the **Start** menu.

1. On the workstation, create the directory `C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs`.

For example: `mkdir C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs`

2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the directory you created on the workstation.
3. On the workstation, make sure the file permissions are set to read-only for `C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem`.
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary server.
5. Install the client tools using guided setup or the command line.

- Guided setup
 - a. Download the Windows [pe-client-tools-package](#).
 - b. Double-click the `pe-client-tools.msi` file.
 - c. Follow prompts to accept the license agreement and select the installation location.
 - d. Click **Install**.
- Command line
 - a. Download the Windows [pe-client-tools-package](#).
 - b. From the command line, run the installer:

```
msiexec /i <PATH TO PE-CLIENT-TOOLS.MSI> TARGETDIR="<INSTALLATION
  DIRECTORY>"
```

TARGETDIR is optional.

Install PE client tools on an unmanaged macOS workstation

You can install the client tools on a macOS workstation using Finder or the command line.

1. On the workstation, create the directory `/etc/puppetlabs/puppet/ssl/certs`.
2. On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the directory you created on the workstation.
3. On the workstation, make sure file permissions are correct: `chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem`
4. Verify that the checksum of `ca.pem` on the workstation matches the checksum of the same file on the primary server.

5. Install the client tools using Finder or the command line.

- Finder
 - a. Download the macOS [pe-client-tools-package](#).
 - b. Open the `pe-client-tools.dmg` and click the installer .pkg.
 - c. Follow the prompts to install the client tools.
- Command line
 - a. Download the macOS [pe-client-tools-package](#).
 - b. Mount the disk image: `sudo hdiutil mount <DMGFILE>`.
 A line appears ending with `/Volumes/puppet-agent-VERSION`. This directory location is the mount point for the virtual volume created from the disk image.
 - c. Run `cd /Volumes/pe-client-tools-VERSION`.
 - d. Run `sudo installer -pkg pe-client-tools-<VERSION>-installer.pkg -target /`.
 - e. Run `cd ~` and then run `sudo umount /Volumes/pe-client-tools-VERSION`.

Configuring and using PE client tools

Use configuration files to customize how client tools communicate with the primary server.

For each client tool, you can create config files for individual machines (global) or for individual users. Configuration files are structured as JSON.

Save configuration files to these locations:

- Global
 - *nix — `/etc/puppetlabs/client-tools/`
 - Windows — `%ProgramData%\puppetlabs\client-tools`
- User
 - *nix — `~/.puppetlabs/client-tools/`
 - Windows — `%USERPROFILE%\puppetlabs\client-tools`

On managed client nodes where the operating system and architecture match the primary server, you can have PE manage Puppet code and orchestrator global configuration files using the `puppet_enterprise::profile::controller` class.

For example configuration files and details about using the various client tools, see the documentation for each service.

Client tool	Documentation
Orchestrator	<ul style="list-style-type: none"> • How Puppet orchestrator works on page 461 • Running Puppet on demand from the CLI on page 482 • Running tasks from the command line on page 497 •
Puppet access	<ul style="list-style-type: none"> • Token-based authentication on page 218
Puppet code	<ul style="list-style-type: none"> • Triggering Code Manager on the command line on page 639

Client tool	Documentation
PuppetDB	<ul style="list-style-type: none"> PuppetDB CLI

Related information

[Orchestrator configuration files](#) on page 472

The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate work station, there are two types of configuration files: a global configuration file and a user-specified configuration file.

[Configure puppet-access](#) on page 218

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. If you want to use `puppet-access`, ensure it is configured correctly before using it to generate authentication tokens.

[Installing and configuring puppet-code](#) on page 639

PE automatically installs and configures the `puppet-code` command on your primary server as part of the included PE client tools package. You can also set up `puppet-code` on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.

Uninstalling

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

Uninstall infrastructure nodes

The `puppet-enterprise-uninstaller` script is installed on the primary server. In order to uninstall, you must run the uninstaller on each infrastructure node.

By default, the uninstaller removes the software, users, logs, cron jobs, and caches, but it leaves your modules, manifests, certificates, databases, and configuration files in place, as well as the home directories of any users it removes. If you want to uninstall and reinstall a primary server on the same system, see [Uninstaller options](#) on page 143 for options that prevent conflicts by removing those files and databases.

1. From the infrastructure node that you want to uninstall, from the command line as root, navigate to the installer directory and run the uninstall command: `$ sudo ./puppet-enterprise-uninstaller`

Note: If you don't have access to the installer directory, you can run `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`.

2. Follow prompts to uninstall.
3. (Optional) If you don't uninstall the primary server, and you plan to reinstall on an infrastructure node at a later date, remove the agent certificate for that component from the primary server. On the primary server: `puppetserver ca clean <PE COMPONENT CERT NAME>`.

Uninstall agents

You can remove the agent from nodes that you no longer want to manage.

Note: Uninstalling the agent doesn't remove the node from your environment. To completely remove all traces of a node, you must also purge the node.

Related information

[Adding and removing agent nodes](#) on page 310

After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.

Uninstall *nix agents

The *nix agent package includes an uninstall script, which you can use when you're ready to retire a node.

1. On the agent node, run the uninstall script: `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`
2. Follow prompts to uninstall.
3. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean <AGENT CERT NAME>`

Uninstall Windows agents

To uninstall the agent from a Windows node, use the Windows **Add or Remove Programs** interface, or uninstall from the command line.

Uninstalling the agent removes the Puppet program directory, the agent service, and all related registry keys. The data directory remains intact, including all SSL keys. To completely remove Puppet from the system, manually delete the data directory.

1. Use the Windows **Add or Remove Programs** interface to remove the agent.

Alternatively, you can uninstall from the command line if you have the original .msi file or know the product code of the installed MSI, for example: `msiexec /qn /norestart /x [puppet.msi|<PRODUCT_CODE>]`

2. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean <AGENT CERT NAME>`

Uninstall macOS agents

Use the command line to remove all aspects of the agent from macOS nodes.

1. On the agent node, run these commands:

```
rm -rf /var/log/puppetlabs
rm -rf /var/run/puppetlabs
pkgutil --forget com.puppetlabs.puppet-agent
launchctl remove puppet
rm -rf /Library/LaunchDaemons/com.puppetlabs.puppet.plist
launchctl remove pxp-agent
rm -rf /Library/LaunchDaemons/com.puppetlabs.pxp-agent.plist
rm -rf /etc/puppetlabs
rm -rf /opt/puppetlabs
```

2. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean <AGENT CERT NAME>`

Uninstaller options

You can use the following command-line flags to change the uninstaller's behavior.

- `-p` — Purge additional files. With this flag, the uninstaller also removes all configuration files, modules, manifests, certificates, the home directories of any users created by the installer, and the Puppet public GPG key used for package verification.
- `-d` — Also remove any databases created during installation.
- `-h` — Display a help message.
- `-n` — Run in noop mode; show commands that would have been run during uninstallation without actually running them.
- `-y` — Don't ask to confirm uninstallation, assuming an answer of yes.

To remove every trace of PE from a system, which is required if you want to reinstall PE on the same system, run:

```
$ sudo ./puppet-enterprise-uninstaller -d -p
```

Upgrading

To upgrade your Puppet Enterprise deployment, you must upgrade both the infrastructure components and agents.

- [Upgrading Puppet Enterprise](#) on page 144
Upgrade your PE installation as new versions become available.
- [Upgrading agents](#) on page 150

Upgrade your agents as new versions of Puppet Enterprise become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can upgrade individual nodes using a script.

Upgrading Puppet Enterprise

Upgrade your PE installation as new versions become available.

Upgrade paths

These are the valid upgrade paths for PE.

If you're on version...	Upgrade to...	Notes
2021.0 (latest)	You're up to date!	
2019.8.z	latest	
2019.y	2019.8.z	
2018.1.2 or later	2019.8.z	You must have version 2018.1.2 or later in order to complete prerequisites for upgrade to 2019.8.z.
2018.1.3 or later with disaster recovery		
2018.1.0 or 2018.1.1	2018.1.z	With disaster recovery enabled, you must have version 2018.1.3 in order to upgrade to 2019.8.z. Alternatively, you can forget and then recreate your replica after upgrade.

Related information

[Component versions in recent PE releases](#) on page 13

These tables show which components are in Puppet Enterprise (PE) releases, covering recent long-term supported (LTS) releases. To see component version tables for a release that has passed its support phase, switch to the docs site for that release.

[Primary server and agent compatibility](#) on page 14

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

Upgrade cautions

These are the major changes to PE since the last long-term support release, 2019.8. Review these recommendations and plan accordingly before upgrading to this version.

Platforms removed in 2021.0

Several agent platforms that were previously deprecated have been removed in PE 2021.0.

Before upgrading to this version, remove the `pe_repo::platform` class for these operating systems from the **PE Master** node group in the console, and from your code and Hiera.

- AIX 6.1
- Enterprise Linux 4
- Enterprise Linux 6, 7 s390x
- Fedora 26, 27, 28, 29
- Mac OS X 10.9, 10.12, 10.13
- SUSE Linux Enterprise Server 11, 12 s390x

Test modules before upgrade

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

Before you begin

If you are already using PDK, your modules should pass validation and unit tests with your currently installed version of PDK.

Update PDK with each new release to ensure compatability with new versions of PE.

1. Download and install PDK. If you already have PDK installed, this updates PDK to its latest version. For detailed instructions and download links, see the [installing](#) instructions.
2. If you have not previously used PDK with your modules, convert them to a PDK compatible format. This makes changes to your module to enable validation and unit testing with PDK. For important usage details, see the [converting modules](#) documentation.

For example, from within the module directory, run:

```
pdk convert
```

3. If your modules are already compatible with PDK, update them to the latest module template. If you converted modules in step 2, you do not need to update the template. To learn more about updating, see the [updating module templates](#) documentation.

For example, from within the module directory, run:

```
pdk update
```

4. Validate and run unit tests for each module, specifying the version of PE you are upgrading to. When specifying a PE version, be sure to specify at least the year and the release number, such as 2018.1. For information about module validations and testing, see the [validating and testing modules](#) documentation.

For example, from within the module directory, run:

```
pdk validate
pdk test unit
```

The `pdk test unit` command verifies that testing dependencies and directories are present and runs the unit tests that you write. It does not create unit tests for your module.

5. If your module fails validation or unit tests, make any necessary changes to your code.

After you've verified that your modules work with the new PE version, you can continue with your upgrade.

Upgrade PE

Upgrade PE infrastructure components to get the latest features and fixes. Follow the upgrade instructions for your installation type to ensure you upgrade components in the correct order.

Before you begin

Review the [upgrade cautions](#) for major changes to architecture and infrastructure components which might affect your upgrade.

Upgrade a standard installation

To upgrade a standard installation, run the PE installer on your primary server, and then upgrade any additional components.

Before you begin

Back up your PE installation.

If you're upgrading a replica, ensure you have a valid admin RBAC token. If you're upgrading **from 2018.1**, the RBAC token must be generated by a user with **Job orchestrator** and **Node group** view permissions.

Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

1. Optional: Speed upgrade by cleaning up PuppetDB reports. On your primary server, run `/opt/puppetlabs/bin/puppetdb delete-reports`
If the command fails to execute, you're likely using a version of PuppetDB that doesn't yet include the command. See [Upgrade cautions](#) on page 145 for manual steps.
2. [Download](#) the tarball appropriate to your operating system and architecture.
3. Unpack the installation tarball: `tar -xf <tarball>`
You need about 1 GB of space to untar the installer.
4. From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:

```
sudo ./puppet-enterprise-installer
```

- If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`

With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

5. Upgrade these additional PE infrastructure components.
 - Agents
 - PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

6. In disaster recovery installations, upgrade your replica.

The replica is temporarily unavailable to serve as backup during this step, so time upgrading your replica to minimize risk.

- a) On your primary server logged in as root, run: `puppet infrastructure upgrade replica <REPLICA_FQDN>`

To specify the location of an authentication token other than the default: `puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file <PATH_TO_TOKEN>`

- b) After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run: `/opt/puppetlabs/bin/puppet-infra status`
- c) If your replica reports errors, reinitialize the replica. On your replica, run: `/opt/puppetlabs/bin/puppet-infra reinitialize replica -y`

7. Optional: Remove old PE packages from your infrastructure nodes: `puppet infrastructure run remove_old_pe_packages pe_version=current`

For `pe_version`, you can specify a SHA, a version number, or `current`. All packages older than the specified version are removed.

Related information

[Back up your infrastructure](#) on page 687

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Upgrade a large installation

To upgrade a large installation, run the PE installer on your primary server, and then upgrade compilers and any additional components.

Before you begin

Back up your PE installation.

Ensure you have a valid admin RBAC token in order to upgrade compilers or a replica. If you're upgrading **from 2018.1**, the RBAC token must be generated by a user with **Job orchestrator** and **Node group** view permissions.

Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

1. Optional: Speed upgrade by cleaning up PuppetDB reports. On your primary server, run `/opt/puppetlabs/bin/puppetdb delete-reports`

If the command fails to execute, you're likely using a version of PuppetDB that doesn't yet include the command. See [Upgrade cautions](#) on page 145 for manual steps.

2. [Download](#) the tarball appropriate to your operating system and architecture.
3. Unpack the installation tarball: `tar -xf <tarball>`

You need about 1 GB of space to untar the installer.

4. From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:

```
sudo ./puppet-enterprise-installer
```

- If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`

With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

5. To upgrade compilers, on your primary server logged in as root, run: `puppet infrastructure upgrade compiler <COMPILER_FQDN-1>, <COMPILER_FQDN-2>`

- To upgrade all compilers simultaneously: `puppet infrastructure upgrade compiler --all`
- To specify the location of an authentication token other than the default: `puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file <PATH_TO_TOKEN>`

6. Upgrade these additional PE infrastructure components.

- Agents
- PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

7. In disaster recovery installations, upgrade your replica.

The replica is temporarily unavailable to serve as backup during this step, so time upgrading your replica to minimize risk.

- a) On your primary server logged in as root, run: `puppet infrastructure upgrade replica <REPLICA_FQDN>`

To specify the location of an authentication token other than the default: `puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file <PATH_TO_TOKEN>`

- b) After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run: `/opt/puppetlabs/bin/puppet-infra status`
- c) If your replica reports errors, reinitialize the replica. On your replica, run: `/opt/puppetlabs/bin/puppet-infra reinitialize replica -y`

8. Optional: Remove old PE packages from your infrastructure nodes: `puppet infrastructure run remove_old_pe_packages pe_version=current`

For `pe_version`, you can specify a SHA, a version number, or `current`. All packages older than the specified version are removed.

Optionally [convert legacy compilers](#) to the new style compiler running the PuppetDB service.

Related information

[Back up your infrastructure](#) on page 687

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Upgrade an extra-large installation

For help upgrading an extra-large installation, reach out to your technical account manager.

Upgrade a standalone PE-PostgreSQL installation

To upgrade a large installation with standalone PE-PostgreSQL, run the PE installer first on your PE-PostgreSQL node, then on your primary server, and then upgrade any additional components.

Before you begin

Back up your PE installation.

Ensure you have a valid admin RBAC token in order to upgrade compilers. If you're upgrading **from 2018.1**, the RBAC token must be generated by a user with **Job orchestrator** and **Node group** view permissions.

Remove from the console (in the **PE Master** node group), Hiera, or `pe.conf` any `agent_version` parameters that you've set in the `pe_repo` class that matches your infrastructure nodes. Doing so ensures that upgrade isn't blocked by attempting to download a non-default agent version for your infrastructure OS and architecture.

- Optional: Speed upgrade by cleaning up PuppetDB reports. On your primary server, run `/opt/puppetlabs/bin/puppetdb delete-reports`
If the command fails to execute, you're likely using a version of PuppetDB that doesn't yet include the command. See [Upgrade cautions](#) on page 145 for manual steps.
- [Download](#) the tarball appropriate to your operating system and architecture.
- Unpack the installation tarball: `tar -xf <tarball>`
You need about 1 GB of space to untar the installer.
- Upgrade your PostgreSQL node.
 - Ensure that the `pe.conf` file on your PostgreSQL node is up to date by running `puppet infrastructure recover_configuration` on your primary server, and then copying `/etc/puppetlabs/enterprise/conf.d` to the PostgreSQL node.
 - Copy the installation tarball to the PostgreSQL node, and from the installer directory, run the installer: `sudo ./puppet-enterprise-installer`
- From the installer directory on your primary server, run the installer and follow the CLI instructions to complete your server upgrade:

```
sudo ./puppet-enterprise-installer
```

- If you want to specify a different `pe.conf` file other than the existing file, use the `-c` flag: `sudo ./puppet-enterprise-installer -c <FULL PATH TO pe.conf>`
With this flag, your previous `pe.conf` is backed up to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and a new `pe.conf` is created at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.
- To upgrade compilers, on your primary server logged in as root, run: `puppet infrastructure upgrade compiler <COMPILER_FQDN-1>, <COMPILER_FQDN-2>`
 - To upgrade all compilers simultaneously: `puppet infrastructure upgrade compiler --all`
 - To specify the location of an authentication token other than the default: `puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file <PATH_TO_TOKEN>`
 - Upgrade these additional PE infrastructure components.
 - Agents
 - PE client tools — On unmanaged nodes only, re-install the version of client tools that matches the PE version you upgraded to. Client tools are automatically updated on infrastructure nodes and managed nodes when you upgrade PE.

8. Optional: Remove old PE packages from your infrastructure nodes: `puppet infrastructure run remove_old_pe_packages pe_version=current`

For `pe_version`, you can specify a SHA, a version number, or `current`. All packages older than the specified version are removed.

Optionally [convert legacy compilers](#) to the new style compiler running the PuppetDB service.

Related information

[Back up your infrastructure](#) on page 687

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Upgrading agents

Upgrade your agents as new versions of Puppet Enterprise become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can upgrade individual nodes using a script.

Note: Before upgrading agents, verify that the primary server and agent software versions are compatible. Then after upgrade, run Puppet on your agents as soon as possible to verify that agents have the correct configuration and that your systems are behaving as expected.

Setting your desired agent version

To upgrade your primary server but use an older agent version that is still compatible with the new primary server, define a `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>` class with the `agent_version` variable set to your desired agent version.

To ensure your agents are always running the same version as your primary server, in the `puppetlabs-puppet_agent` module, set the `package_version` variable for the `puppet_agent` class to `auto`. This causes agents to automatically upgrade themselves on their first Puppet run after a primary server upgrade.

Related information

[Upgrading Puppet Enterprise](#) on page 144

Upgrade your PE installation as new versions become available.

Upgrade agents using the puppet_agent module

The `puppetlabs-puppet_agent` module, available from the Forge, enables you to upgrade multiple *nix or Windows agents at one time. The module handles all the latest version-to-version upgrades.

Important: For the most reliable upgrade, always use the latest version of the `puppet_agent` module to upgrade agents. Test the upgrade on a subset of agents, and after you verify the upgrade, upgrade remaining agents.

1. On your primary server, download and install the `puppetlabs-puppet_agent` module: `puppet module install puppetlabs-puppet_agent`

2. Configure the primary server to download the agent version you want to upgrade.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - b) On the **Classes** tab, in the **Add a new class** field, enter `pe_repo`, and select the appropriate repo class from the list of classes.

Repo classes are listed as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

To specify a particular agent version, set the `agent_version` variable using an X.Y.Z format (for example, 5.5.14). When their version is set explicitly, agents do not automatically upgrade when you upgrade your primary server.

- c) Click **Add class** and commit changes.
- d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`

The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>/`.

3. Click **Node groups**, click **Add group**, specify options for a new upgrade node group, and then click **Add**.
 - **Parent name** — Select the name of the classification node group that you want to set as the parent to this group, in this case, **All Nodes**.
 - **Group name** — Enter a name that describes the role of this classification node group, for example, `agent_upgrade`.
 - **Environment** — Select the environment your agents are in.
 - **Environment group** — *Do not* select this option.
4. Click the link to **Add membership rules, classes, and variables**.
5. On the **Rules** tab, create a rule to add the agents that you want to upgrade to this group, click **Add Rule**, and then commit changes.

For example:

- **Fact** — `osfamily`
- **Operator** — `=`
- **Value** — `RedHat`

6. Still in the agent upgrade group, on the **Classes** tab, add the **puppet_agent** class, and click **Add class**.

If you don't immediately see the class, click **Refresh** to update the classifier.

Note: If you've changed the prefix parameter of the **pe_repo** class in your **PE Master** node group, set the **puppet-agent source** parameter of the upgrade group to `https://<PRIMARY_HOSTNAME>:8140/<Prefix>`.

7. In the **puppet_agent** class, specify the version of the `puppet-agent` package version that you want to install, then commit changes.

Parameter	Value
package_version	<p>The <code>puppet-agent</code> package version to install, for example 5.3.3.</p> <p>Set this parameter to <code>auto</code> to install the same agent version that is installed on your primary server.</p>

8. On the agents that you're upgrading, run Puppet: `/opt/puppet/bin/puppet agent -t`

After the Puppet run, you can verify the upgrade with `/opt/puppetlabs/bin/puppet --version`

Upgrade agents using a script

To upgrade an individual node, for example to test or troubleshoot, you can upgrade directly from the node using a script. This method relies on a package repository hosted on your primary server.

Note: If you encounter SSL errors during the upgrade process, ensure your agent's OpenSSL is up to date and matches the primary server's version. You can check the primary server's OpenSSL versions with `/opt/puppetlabs/puppet/bin/openssl version` and the agent's version with `openssl version`.

Upgrade a *nix agent using a script

You can upgrade an individual *nix agent using a script.

1. Configure the primary server to download the agent version you want to upgrade.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - b) On the **Classes** tab, in the **Add a new class** field, enter `pe_repo`, and select the appropriate repo class from the list of classes.

Repo classes are listed as `pe_repo::platform:<AGENT_OS_VERSION_ARCHITECTURE>`.

To specify a particular agent version, set the `agent_version` variable using an X.Y.Z format (for example, 5.5.14). When their version is set explicitly, agents do not automatically upgrade when you upgrade your primary server.

- c) Click **Add class** and commit changes.
- d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`

The new repo is created in `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>/`.

2. SSH into the agent node you want to upgrade.
3. Run the upgrade command appropriate to the operating system.

- Most *nix

```
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8140/packages/current/
install.bash"

curl --cacert "$cacert" "$uri" | sudo bash
```

See [Usage notes for curl examples](#) for information about forming curl commands.

- Mac OS X, Solaris, and AIX

```
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8140/packages/current/
install.bash"

curl --cacert "$cacert" "$uri" | sudo bash
```

PE services restarts automatically after upgrade.

Upgrade a Windows agent using a script

You can upgrade an individual Windows agent using a script. For Windows, this method is riskier than using the `puppet_agent` module to upgrade, because you must manually complete and verify steps that the module handles automatically.

Note: The `<PRIMARY_HOSTNAME>` portion of the installer script—as provided in the following example—refers to the FQDN of the primary server. The FQDN must be fully resolvable by the machine on which you're installing or upgrading the agent.

1. Stop the Puppet service and the PXP agent service.

2. On the Windows agent, run the install script:

```
[Net.ServicePointManager]::ServerCertificateValidationCallback
= {$true}; $webClient = New-Object System.Net.WebClient;
$webClient.DownloadFile('https://<PRIMARY_HOSTNAME>:8140/packages/
current/install.ps1', 'install.ps1'); .\install.ps1
```

3. Verify that Puppet runs are complete.
4. Restart the Puppet service and the PXP agent service.

Upgrade agents without internet access

If you don't have access to the internet beyond your infrastructure you can download the appropriate agent tarball from an internet-connected system and then upgrade using a script.

Before you begin

[Download](#) the appropriate agent tarball from an internet-connected system.

1. On your primary server, copy the agent tarball to `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/<PLATFORM>`.
2. Run Puppet: `puppet agent -t`
3. Follow the steps to [Upgrade agents using a script](#) on page 152.

Upgrading the agent independent of PE

You can optionally upgrade the agent to a newer version than the one packaged with your current PE installation.

For details about Puppet agents versions that are tested and supported for PE, see the PE component version table.

The agent version is specified on a platform-by-platform basis in the **PE Master** node group, in any `pe_repo::platform` class, using the `agent_version` parameter.

When you install new nodes or upgrade existing nodes, the agent install script installs the version of the agent specified for its platform class. If a version isn't specified for the node's platform, the script installs the default version packaged with your current version of PE.

Note: To install nodes without internet access, download the agent tarball for the version you want to install, as specified using the `agent_version` parameter.

The platform in use on your primary server requires special consideration. The agent version used on your primary server must match the agent version used on other infrastructure nodes, including compilers and replicas, otherwise your primary server won't compile catalogs for these nodes.

To keep infrastructure nodes synced to the same agent version, if you specify a newer `agent_version` for your primary server platform, you must either:

- (Recommended) Upgrade the agent on your primary server—and any existing infrastructure nodes—to the newer agent version. You can upgrade these nodes by running the agent install script.
- Manually install the older agent version used on your primary server on any new infrastructure nodes you provision. You **can't** install these nodes using the agent install script, because the script uses the agent version specified for the platform class, instead of the primary server's current agent version. Manual installation requires configuring `puppet.conf`, DNS alt names, CSR attributes, and other relevant settings.

Related information

[Component versions in recent PE releases](#) on page 13

These tables show which components are in Puppet Enterprise (PE) releases, covering recent long-term supported (LTS) releases. To see component version tables for a release that has passed its support phase, switch to the docs site for that release.

Configuring Puppet Enterprise

- [Tuning infrastructure nodes](#) on page 154

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

- [Methods for configuring PE](#) on page 160

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

- [Configuring Puppet Server](#) on page 163

After you've installed Puppet Enterprise, optimize it for your environment by configuring Puppet Server settings as needed.

- [Configuring PuppetDB](#) on page 166

After you've installed Puppet Enterprise, optimize it for your environment by configuring PuppetDB as needed.

- [Configuring security settings](#) on page 167

Ensure your PE environment is secure by configuring security settings.

- [Configuring proxies](#) on page 169

You can work around limited internet access by configuring proxies at various points in your infrastructure, depending on your connectivity limitations.

- [Configuring the console](#) on page 171

After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.

- [Configuring orchestration](#) on page 173

After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.

- [Configuring ulimit](#) on page 176

As your infrastructure grows and you bring more agents under management, you might need to increase the number allowed file handles per client.

- [Writing configuration files](#) on page 177

Puppet supports two formats for configuration files that configure settings: valid JSON and Human-Optimized Config Object Notation (HOCON), a JSON superset.

- [Analytics data collection](#) on page 178

Some components automatically collect data about how you use Puppet Enterprise. If you want to opt out of providing this data, you can do so, either during or after installing.

- [Static catalogs](#) on page 182

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing `source` attributes pointing to `puppet:///` locations on a node.

Tuning infrastructure nodes

Use these guidelines to configure your installation to maximize its use of available system (CPU and RAM) resources.

PE is composed of multiple services on one or more infrastructure hosts. Each service has multiple settings that can be configured to maximize use of system resources and optimize performance. The default settings for each service

are conservative, because the set of services sharing resources on each host varies depending on your infrastructure. Optimized settings vary depending on the complexity and scale of your infrastructure.

Configure settings after an install or upgrade, or after making changes to infrastructure hosts, including changing the system resources of existing hosts, or adding new hosts, including compilers.

Related information

[Hardware requirements](#) on page 81

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

Primary server tuning

These are the default and recommended tuning settings for your primary server or disaster recovery replica.

Note: Recommended settings are appropriate for standard installations or large installations with compilers running the PuppetDB service. Installations with legacy compilers generally require more resources on the primary server for PuppetDB.

		Puppet Server			PuppetDB		Console	Orchestrator		PostgreSQL	
		JRuby max active instances	Java heap (MB)	Reserved code cache (MB)	Command processing threads	Java heap (MB)	Java heap (MB)	Java heap (MB)	JRuby max active instances	Shared buffers (MB)	Work memory (MB)
4 cores, 8 GB RAM	Default	3	2048	512	2	256	256	704	1	976	4
	Recommended	2	1024	192	1	819	655	819	1	1638	4
	With legacy compilers	2	1024	192	2	1228	655	819	1	1638	4
6 cores, 10 GB RAM	Default	4	2048	512	3	256	256	704	1	1488	4
	Recommended	3	2304	288	1	1024	819	1024	1	2048	4
	With legacy compilers	2	1536	192	3	1536	819	1024	1	2048	4
8 cores, 12 GB RAM	Default	4	2048	512	4	256	256	704	1	2000	4
	Recommended	3	2304	288	2	1228	983	1228	1	2457	4
	With legacy compilers	3	2304	288	4	1843	983	1228	1	2457	4
10 cores, 16 GB RAM	Default	4	2048	512	5	256	256	704	1	3024	4
	Recommended	3	3840	480	2	1638	1024	1638	2	3276	4
	With legacy compilers	4	3072	384	5	2457	1024	1638	2	3276	4
12 cores, 24GB RAM	Default	4	2048	512	6	256	256	704	1	4096	4
	Recommended	3	6144	768	3	2457	1024	2457	3	4915	4
	With legacy compilers	5	3840	480	6	3686	1024	2457	3	4915	4

16 cores, 32GB RAM	Default	4	2048	512	8	256	256	704	1	4096	4
	Recommended	4	9216	864	4	3276	1024	3276	3	6553	4
	With legacy compilers	7	7168	672	8	4915	1024	3276	3	6553	4

Compiler tuning

These are the default and recommended tuning settings for compilers running the PuppetDB service.

		Puppet Server				PuppetDB			
		JRuby max active instances	Java heap (MB)	Reserved code cache (MB)	Command processing threads	Java heap (MB)	Read Maximum Pool Size	Write Maximum Pool Size	
4 cores, 8 GB RAM	Default	3	1536	384	1	819	4	2	
6 cores, 12 GB RAM	Default	4	2048	512	1	1228	6	2	
	Recommended	4	3072	512	1	1228	6	2	

Legacy compiler tuning

These are the default and recommended tuning settings for legacy compilers without the PuppetDB service.

		Puppet Server		
		JRuby max active instances	Java heap (MB)	Reserved code cache (MB)
4 cores, 8 GB RAM	Default	3	2048	512
6 cores, 12 GB RAM	Default	4	2048	512
	Recommended	5	3840	480

Using the `puppet infrastructure tune` command

The `puppet infrastructure tune` command outputs optimized settings for PE services based on recommended guidelines.

When you run `puppet infrastructure tune`, it queries PuppetDB to identify infrastructure hosts and their processor and memory facts, and outputs settings in YAML format for use in Hiera.

The command is compatible with most standard PE configurations, including those with compilers, a replica, or standalone PE-PostgreSQL. The command must be run on your primary server as root.

These are the options commonly used with the `puppet infrastructure tune` command:

- `--current` outputs existing tuning settings from the console and Hiera, and identifies duplicate settings found in both places.
- `--memory_per_jruby <MB>` outputs tuning recommendations based on specified memory allocated to each JRuby in Puppet Server. If you implement tuning recommendations using this option, specify the same value for `puppetserver_ram_per_jruby`.
- `--memory_reserved_for_os <MB>` outputs tuning recommendations based on specified RAM reserved for the operating system.
- `--common` outputs common settings — identical on several nodes — separately from node-specific settings.

For more information about the `tune` command, run `puppet infrastructure tune --help`.

Related information

[RAM per JRuby](#) on page 157

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server. In installations with compilers running the PuppetDB service, this setting is a good starting point for tuning your installation, because the value you specify is factored into several other parameters, including JRuby max active instances and heap allocation on compilers running PuppetDB.

Tuning parameters

Tuning parameters let you customize PE components for maximum performance and hardware resource utilization.

Specify tuning parameters using Hiera for the best scalability and consistency. If you must use the console, add the parameter to the appropriate infrastructure node group using the method suitable for the parameter type:

- Specify `puppet_enterprise::profile` parameters, including `java_args`, `shared_buffers`, and `work_mem`, as parameters of their class.
- Specify all other tuning parameters as configuration data.

RAM per JRuby

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server. In installations with compilers running the PuppetDB service, this setting is a good starting point for tuning your installation, because the value you specify is factored into several other parameters, including JRuby max active instances and heap allocation on compilers running PuppetDB.

Parameter

```
puppet_enterprise::puppetserver_ram_per_jruby
```

Default value

512 MB

Accepted values

Integer (MB)

How to calculate

If you have complex Hiera code, many environments or modules, or large reports, you might need to increase this setting. You can generally achieve good performance by allocating up to around 2 GB per JRuby. If 2 GB is inadequate, you might benefit from enabling environment caching.

Console node group

PE Master

JRuby max active instances

The `jruby_max_active_instances` setting controls the maximum number of JRuby instances to allow on the Puppet Server.

Parameter

```
puppet_enterprise::master::puppetserver::jruby_max_active_instances
```

This setting is referred to as `max_active_instances` in the `pe-puppet-server.conf` file and in open source Puppet. It's the same setting.

Default value

Primary server — Number of CPUs - 1, minimum 1, maximum 4

Compilers — Number of CPUs x 0.75, minimum 1, maximum 24

Accepted values

Integer

How to calculate

As a conservative estimate, one JRuby process uses approximately 512 MB of RAM. Four JRuby instances works for most environments. Because increasing the maximum number of JRuby instances also increases the amount of RAM used by Puppet Server, make sure the Puppet Server heap size (`java_args`) is scaled proportionally. For example, if you set `jruby_max_active_instances` to 4, you should set Puppet Server `java_args` to at least 2 GB.

Console node group

PE Master or, for compilers running the PuppetDB service, PE Compiler

JRuby max requests per instance

The `jruby_max_requests_per_instance` setting determines the maximum number of HTTP requests a JRuby handles before it's terminated. When a JRuby instance reaches this limit, it's flushed from memory and replaced with a fresh one.

Parameter

```
puppet_enterprise::master::puppetserver::jruby_max_requests_per_instance
```

This setting is referred to as `max_requests_per_instance` in the `pe-puppet-server.conf` file and in open source Puppet. It's the same setting.

Default value

100,000

Accepted values

Integer

How to calculate

More frequent JRuby flushing can help address memory leaks, because it prevents any one interpreter from consuming too much RAM. However, performance is reduced slightly each time a new JRuby instance loads. Ideally, set this parameter to get a new interpreter no more than every few hours. There are multiple interpreters running with requests balanced across them, so the lifespan of each interpreter varies.

Console node group

PE Master

Puppet Server reserved code cache

The `reserved_code_cache` setting specifies the maximum space available to store the Puppet Server code cache during catalog compilation.

Parameter

```
puppet_enterprise::master::puppetserver::reserved_code_cache
```

Default value

Primary server — If total RAM is less than 2 GB, the Java default is used. Otherwise, 512 MB.

Compilers — Number of JRuby instances x 128 MB, min 128 MB, max 2048 MB

Accepted values

Integer (MB)

How to calculate

JRuby requires an estimated 128 MB of cache space per instance, so to determine the minimum amount of space needed: `number of JRuby instances x 128 MB`

Console node group

PE Master or, for compilers running the PuppetDB service, PE Compiler

Java heap

The `java_args` setting is used to specify *heap size*: the amount of memory that each Java process is allowed to request from the operating system. You can specify heap size for each PE service that uses Java, including Puppet Server, PuppetDB, and console and orchestration services.

Heap size is specified as `Xmx` and `Xms`, the maximum and minimum heap size, respectively. Typically, the maximum and minimum are set to the same value so that heap size is fixed, for example `{ 'Xmx' => '2048m', 'Xms' => '2048m' }`.

Parameters

Puppet Server — `puppet_enterprise::profile::master::java_args`

Tip: This setting might be referred to as `puppet_enterprise::master::java_args` or `puppet_enterprise::master::puppetserver::java_args`. They are all the same thing: `profile::master` filters down to `master`, which filters down to `master::puppetserver`.

PuppetDB — `puppet_enterprise::profile::puppetdb`

Console services — `puppet_enterprise::profile::console`

Orchestration services — `puppet_enterprise::profile::orchestrator`

Default values

Puppet Server — 2 GB

PuppetDB — 256 MB

Console services — 256 MB

Orchestration services — 704 MB

Accepted values

JSON string

Console node group

Puppet Server — PE Master or PE Compiler

PuppetDB — PE PuppetDB or, for compilers running the PuppetDB service, PE Compiler

Console services — PE Console

Orchestration services — PE Orchestrator

PuppetDB command processing threads

The `command_processing_threads` setting specifies how many command processing threads PuppetDB uses to sort incoming data. Each thread can process a single command at a time.

Parameter

`puppet_enterprise::puppetdb::command_processing_threads`

Default value

Primary server — Number of CPUs x 0.5, minimum 1

Compilers — Number of CPUs x 0.25, minimum 1, maximum 3

Accepted values

Integer

How to calculate

If the PuppetDB queue is backing up and you have CPU cores to spare, increasing the number of threads can help process the backlog more rapidly. Avoid allocating all of your CPU cores for command processing, as doing so can starve other PuppetDB subsystems of resources and actually decrease throughput.

Console node group

PE PuppetDB or, for compilers running the PuppetDB service, PE Compiler

PostgreSQL shared buffers

The `shared_buffers` setting specifies the amount of memory the PE-PostgreSQL server uses for shared memory buffers.

Parameter

`puppet_enterprise::profile::database::shared_buffers`

Default value

Available RAM x 0.25, minimum 32 MB, maximum 4096 MB

Accepted values

Integer (MB)

How to calculate

The default of 25 percent of available RAM is suitable for most installations, but you might see improved console performance by increasing `shared_buffers` up to 40 percent of available RAM.

Console node group

PE Database

PostgreSQL working memory

The `work_mem` setting specifies the maximum amount of memory used for queries before writing to temporary files.

Parameter

`puppet_enterprise::profile::database::work_mem`

Default value

$(\text{Available RAM} / 1024 / 8) + 0.5$, minimum 4, maximum 16

Accepted values

Integer (MB)

Console node group

PE Database

Methods for configuring PE

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

PE shares configuration settings used in open source Puppet and documented in the [Configuration Reference](#), however PE defaults for certain settings might differ from the Puppet defaults. Some examples of settings that have different PE defaults include `disable18n`, `environment_timeout`, `always_retry_plugins`, and the Puppet Server JRuby `max-active-instances` setting. To verify PE configuration defaults, check the `puppet.conf` file after installation.

There are three main methods for configuring PE: using the console, adding a key to Hiera, or editing `pe.conf`. Be consistent with the method you choose, unless the situation calls for you to use a specific method over the others.

Configure settings using the console

The console allows you to use a graphical interface to configure Puppet Enterprise (PE).

Changes in the console will override your Hiera data and data in `pe.conf`. It is usually best to use the console when you want to:

- Change parameters in profile classes starting with `puppet_enterprise::profile`.
- Add any parameters in PE-managed configuration files.
- Set parameters that configure at runtime.

There are two ways to change settings in the console: setting configuration data and editing parameters.

Related information

[Preconfigured node groups](#) on page 330

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

Set configuration data

Configuration data set in the console is used for automatic parameter lookup, the same way that Hiera data is used. Console configuration data takes precedence over Hiera data, but you can combine data from both sources to configure nodes.

Tip: In most cases, setting configuration data in Hiera is the more scalable and consistent method, but there are some cases where the console is preferable. Use the console to set configuration data if:

- You want to override Hiera data. Data set in the console overrides Hiera data when configured as recommended.
- You want to give someone access to set or change data and they don't have the skill set to do it in Hiera.
- You simply prefer the console user interface.

Important: If your installation includes a disaster recovery replica, make sure you've correctly enabled data editing in the console for both your primary server and replica.

1. In the console, click **Node groups**, then find the node group that you want to add configuration data to and select it.
2. On the **Configuration data** tab, specify a **Class** and select a **Parameter** to add.

You can select from existing classes and parameters in the node group's environment, or you can specify free-form values. Classes aren't validated, but any class you specify must be present in the node's catalog at runtime in order for the parameter value to be applied.

When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. Optional: Change the default parameter **Value**.

Related information

[Enable data editing in the console](#) on page 173

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

Set parameters

Parameters are declared resource-style, which means they can be used to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

1. In the console, click **Node groups**, and then find the node group that you want to add a parameter to and select it.
2. On the **Classes** tab, select the class you want to modify and the **Parameter** to add.

The **Parameter** drop-down list shows all of the parameters that are available for that class in the node group's environment. When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. (Optional) Change the default **Value**.

Configure settings with Hiera

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

Before you begin

For more information on how to use Hiera, see the [Hiera docs](#).

Changes to PE configuration in Hiera will override configuration settings in `pe.conf`, but not those set in the console. It's best to use Hiera when you want to:

- Change parameters in non-profile classes.
- Set parameters that are static and version controlled.
- Configure for high availability.

To configure a setting using Hiera:

1. Open your default data file.

The default location for Hiera data files is:

- *nix: `/etc/puppetlabs/code/environments/<ENVIRONMENT>/data/common.yaml`
- Windows: `%CommonAppData%\PuppetLabs\code\environments\<ENVIRONMENT>\data\common.yaml`

If you customize the `hiera.yaml` configuration to change location for data files (the `datadir` setting) or the path of the common data file (in the `hierarchy` section), look for the default `.yaml` file in the customized location.

2. Add your new parameter to the file in editor.

For example, to increase the number of seconds before a node is considered unresponsive from the default 3600 to 4000, add the following to your `.yaml` default file and insert your new parameter at the end.

```
Puppet_enterprise::console_services::no_longer_reporting_cutoff: 4000
```

3. To compile changes, run `puppet agent -t`

Related information

[Preconfigured node groups](#) on page 330

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

Configure settings in `pe.conf`

Puppet Enterprise (PE) configuration data includes any data set in `/etc/puppetlabs/enterprise/conf.d/` but `pe.conf` is the file used for most configuration activities during installation.

PE configuration settings made in Hiera and the console always override settings made in `pe.conf`. Configure settings using `pe.conf` when you want to:

- Access settings during installation.
- Configure for high availability.

To configure settings using `pe.conf`:

1. Open the `pe.conf` file on your primary server:

```
/etc/puppetlabs/enterprise/conf.d/pe.conf
```

2. Add the parameter and new value you want to set.

For example, to change the proxy in your repo, add the following and change the parameter to your new proxy location.

```
pe_repo::http_proxy_host: "proxy.example.vlan"
```

3. Run `puppet agent -t`

Note: If PE services are stopped, run `puppet infrastructure configure` instead of `puppet agent -t`.

Configuring Puppet Server

After you've installed Puppet Enterprise, optimize it for your environment by configuring Puppet Server settings as needed.

Tune the Ruby load path

The `ruby_load_path` setting determines where Puppet Server finds components such as Puppet and Facter.

The default setting is located at `$puppetserver_jruby_puppet_ruby_load_path = ['/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib']`.

Note: If you change the `libdir` you must also change the `vardir`.

To change the path to a different array in `pe.conf`:

1. Add the following to your `pe.conf` file on your primary server and set your new load path parameter.

```
puppet_enterprise::master::puppetserver::puppetserver_jruby_puppet_ruby_load_path
```

2. Run `puppet agent -t`

Tune the multithreaded server setting

The `jruby_puppet_multithreaded` setting allows you to enable multithreaded mode, which uses a single JRuby instance to process requests, like catalog compiles, concurrently. The setting defaults to `false`.

To enable or disable the setting using Hiera:

1. Add the following code to your default `.yaml` file and set the parameter to `true` (enable) or `false` (disable). For example:

```
puppet_enterprise::master::puppetserver::jruby_puppet_multithreaded: <true
or false>
```

2. To compile the changes, run `puppet agent -t`

Enable or disable cached data when updating classes

The optional `environment-class-cache-enabled` setting specifies whether cached data is used when updating classes in the console. When `true`, Puppet Server refreshes classes using file sync, improving performance.

The default value for `environment-class-cache-enabled` depends on whether you use Code Manager.

- With Code Manager, the default value is enabled (`true`). File sync clears the cache automatically in the background, so clearing the environment cache manually isn't required when using Code Manager.
- Without Code Manager, the default value is disabled (`false`).

Note: If you're not using Code Manager and opt to enable this setting, make sure your code deployment method — for example `r10k` — clears the environment cache when it completes. If you don't clear the environment cache, the Node Classifier doesn't receive new class information until Puppet Server is restarted.

To enable or disable the cache using Hiera:

1. Add the following code to your default `.yaml` file and set the parameter to the appropriate setting.

```
puppet_enterprise::master::puppetserver::
  jruby_environment_class_cache_enabled: <true OR false>
```

2. To compile the changes, run `puppet agent -t`

Change the `environment_timeout` setting

The `environment_timeout` setting controls how long the primary server caches data it loads from an environment, determining how much time passes before changes to an environment's Puppet code are reflected in its environment.

In PE, the `environment_timeout` is set to 0. This lowers the performance of your primary server but makes it easy for new users to deploy updated Puppet code. Once your code deployment process is mature, change this setting to unlimited.

Note: When you install Code Manager and set the `code_manager_auto_configure` parameter to `true`, `environment_timeout` is updated to unlimited.

To change the `environment_timeout` setting using `pe.conf`:

1. Add the following to your `pe.conf` file on your primary server and specify either 0 or unlimited:

```
puppet_enterprise::master::environment_timeout:<time>
```

2. Run `puppet agent -t`

For more information, see [Environments limitations](#)

Add certificates to the `puppet-admin` certificate allowlist

Add trusted certificates to the `puppet-admin` certificate allowlist.

To add allowed certificates using `pe.conf`:

1. Add the following code to your `pe.conf` file on your primary server and add the desired certificates.

```
puppet_enterprise::master::puppetserver::puppet_admin_certs:'example_cert_name'
```

2. Run `puppet agent -t`

Disable update checking

Puppet Server (pe-puppetserver) checks for updates when it starts or restarts, and every 24 hours thereafter. It transmits basic, anonymous info to our servers at Puppet, Inc. to get update information. You can optionally turn this off.

Specifically, it transmits:

- Product name
- Puppet Server version
- IP address
- Data collection timestamp

To turn off update checking using the console:

1. Open the console, click **Node groups**, and select the **PE Master** node group.
2. On the **Classes** tab, find the `puppet_enterprise::profile::master` class, add the `check_for_updates` parameter from the list, and change its value to `false`.
3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Puppet Server configuration files

At startup, Puppet Server reads all of the `.conf` files in the `conf.d` directory (`/etc/puppetlabs/puppetserver/conf.d`).

The `conf.d` directory contains the following files:

File name	Description
<code>auth.conf</code>	Contains authentication rules and settings for agents and API endpoint access.
<code>global.conf</code>	Contains global configuration settings for Puppet Server, including logging settings.
<code>metrics.conf</code>	Contains settings for Puppet Server metrics services.
<code>pe-puppet-server.conf</code>	Contains Puppet Server settings specific to Puppet Enterprise.
<code>webserver.conf</code>	Contains SSL service configuration settings.
<code>ca.conf</code>	(Deprecated) Contains rules for Certificate Authority services. Superseded by <code>webserver.conf</code> and <code>auth.conf</code> .

For information about Puppet Server configuration files, see [Puppet Server's config files](#) for the Puppet Server version you're using, and the Related information links below.

Related information

[Viewing and managing Puppet Server metrics](#) on page 287

Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

pe-puppet-server.conf settings

The `pe-puppet-server.conf` file contains Puppet Server settings specific to Puppet Enterprise, with all settings wrapped in a `ruby-puppet` section.

gen-home

Determines where JRuby looks for gems. It is also used by the `puppetserver gem` command line tool.

Default: `/opt/puppetlabs/puppet/cache/jruby-gems`

master-conf-dir

Sets the Puppet configuration directory's path.

Default: `/etc/puppetlabs/puppet`

master-var-dir

Sets the Puppet variable directory's path.

Default: `/opt/puppetlabs/server/data/puppetserver`

max-queued-requests

Optional. Sets the maximum number of requests that can be queued waiting to borrow a from the pool. After this limit is exceeded, a `503 Service Unavailable` response is returned for all new requests until the queue drops below the limit.

If `max-retry-delay` is set to a positive value, then the 503 response includes a `Retry-After` header indicating a random sleep time after which the client can retry the request.

Note: Don't use this solution if your managed infrastructure includes a significant number of agents older than Puppet 5.3. Older agents treat a 503 response as a failure, which ends their runs, causing groups of older agents to schedule their next runs at the same time, creating a thundering herd problem.

Default: 0

max-retry-delay

Optional. Sets the upper limit in seconds for the random sleep set as a `Retry-After` header on 503 responses returned when `max-queued-requests` is enabled.

Default: 1800

jruby_max_active_instances

Controls the maximum number of JRuby instances to allow on the Puppet Server.

Default: 4

max_requests_per_instance

Sets the maximum number of requests per instance of a JRuby interpreter before it is killed.

Default: 100000

ruby-load-path

Sets the Puppet configuration directory's path. The agent's `libdir` value is added by default.

Default: `'/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', '/opt/puppetlabs/puppet/cache/lib'`

Configuring PuppetDB

After you've installed Puppet Enterprise, optimize it for your environment by configuring PuppetDB as needed.

This page covers a few key topics, but additional settings and information about configuring PuppetDB is available in the [PuppetDB configuration documentation](#). Be sure to check that the PuppetDB docs version you're looking at matches the one version of PuppetDB in your PE.

Configure agent run reports

By default, every time Puppet runs, the primary server generates agent run reports and submits them to PuppetDB. You can enable or disable this as needed.

To enable or disable agent run reports using the console:

1. Click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
2. On the **Classes** tab, add the `puppet_enterprise::profile::master::puppetdb` class, select the `report_processor_ensure` parameter, and enter the value `present` to enable agent run reports or `absent` to disable agent run reports.
3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Configure how long before PE stops managing deactivated nodes

Use the `node-purge-ttl` parameter to set the "length of time" value before PE automatically removes nodes that have been deactivated or expired. This also removes all facts, catalogs, and reports for the relevant nodes.

To change the amount of time before nodes are purged using the console:

1. Click **Node groups**, and in the PE Infrastructure group, select the **PE Database** group.

2. On the **Classes** tab, find the `puppet_enterprise::profile::puppetdb` class, find the `node_purge_ttl` parameter, and change its value to the desired amount of time.

To change the unit of time, use the following suffixes:

- d - days
- h - hours
- m - minutes
- s - seconds
- ms - milliseconds

For example, to set the purge time to 14 days:

```
puppet_enterprise::profile::puppetdb::node_purge_ttl: '14d'
```

3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Change the PuppetDB user password

The console uses a database user account to access its PostgreSQL database. Change it if it is compromised or to comply with security guidelines.

To change the password:

1. Stop the `pe-puppetdb` puppet service by running `puppet resource service pe-puppetdb ensure=stopped`
2. On the database server (which might or might not be the same as PuppetDB, depending on your deployment's architecture), use the PostgreSQL administration tool of your choice to change the user's password. With the standard PostgreSQL client, you can do this by running `ALTER USER console PASSWORD '<new password>';`
3. Edit `/etc/puppetlabs/puppetdb/conf.d/database.ini` on the PuppetDB server and change the `password:` line under `common` or `production`, depending on your configuration, to contain the new password.
4. Start the `pe-puppetdb` service on the console server by running `puppet resource service pe-puppetdb ensure=running`

Configure excluded facts

Use the `facts_blacklist` parameter exclude facts from being stored in the PuppetDB database.

To specify which facts you want to exclude using Hiera:

1. Add the following to you default `.yaml` file and list the facts you want to exclude. For example, to exclude the facts `system_uptime_example` and `mountpoints_example`:

```
puppet_enterprise::puppetdb::database_ini::facts_blacklist:
- 'system_uptime_example'
- 'mountpoints_example'
```

2. To compile changes, run `puppet agent -t`

Configuring security settings

Ensure your PE environment is secure by configuring security settings.

Configure cipher suites

Due to regulatory compliance or other security requirements, you may need to change which cipher suites your SSL-enabled PE services use to communicate with other PE components.

SSL ciphers for core Puppet services

To add or remove cipher suites for [core Puppet services](#), use Hiera to add an array of SSL ciphers to the `puppet_enterprise::ssl_cipher_suites` parameter.

Note: Changing this parameter overrides the default list of SSL cipher suites.

The example Hiera data below replaces the default list of cipher suites to only allow the four specified.

```
puppet_enterprise::ssl_cipher_suites:
- 'SSL_RSA_WITH_NULL_MD5'
- 'SSL_RSA_WITH_NULL_SHA'
- 'TLS_DH_anon_WITH_AES_128_CBC_SHA'
- 'TLS_DH_anon_WITH_AES_128_CBC_SHA256'
```

Note: Cipher names are in [IANA RFC naming format](#).

SSL for console services

To add or remove cipher suites for console services affecting traffic on port 443, use Hiera or the console to change the `puppet_enterprise::profile::console::proxy::ssl_ciphers` parameter.

For example, to change the parameter in the console, in the **PE Console** node group, add an array of SSL ciphers to the `ssl_ciphers` parameter in the `puppet_enterprise::profile::console::proxy` class.

Configure SSL protocols

Add or remove SSL protocols in your PE infrastructure.

To change what SSL protocols your PE infrastructure uses, use Hiera or the console to add or remove protocols.

Use the parameter `puppet_enterprise::master::puppetserver::ssl_protocols` and add an array for protocols you want to include, or remove protocols you no longer want to use.

For example, to enable TLSv1.1 and TLSv1.2, set the following parameter in the **PE Infrastructure** group in the console or in your Hiera data.

```
puppet_enterprise::master::puppetserver::ssl_protocols[ "TLSv1.1", "TLSv1.2" ]
```

Note: To comply with security regulations, PE 2019.1 and later uses only version 1.2 of the Transport Layer Security (TLS) protocol.

Configure RBAC and token-based authentication settings

Tune RBAC and token-based authentication settings, like setting the number of failed attempts a user has before they are locked out of the console or changing the amount of time a token is valid for.

RBAC and token authentication settings can be changed in the **PE Infrastructure** group in the console or in your Hiera data. Below is a list of related settings.

`puppet_enterprise::profile::console::rbac_failed_attempts_lockout`

An integer specifying how many failed login attempts are allowed on an account before the account is revoked. The default is "10" (attempts).

`puppet_enterprise::profile::console::rbac_password_reset_expiration`

An integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user so that they can reset their password. The default is "24" (hours).

puppet_enterprise::profile::console::rbac_session_timeout

Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console. The default is "60" (minutes).

puppet_enterprise::profile::console::rbac_token_auth_lifetime

A value representing the default authentication lifetime for a token. It cannot exceed the `rbac_token_maximum_lifetime`. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "1h".

puppet_enterprise::profile::console::rbac_token_maximum_lifetime

A value representing the maximum allowable lifetime for all tokens. This is represented as a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). The default is "10y".

puppet_enterprise::profile::console::rbac_account_expiry_check_minutes

An integer that specifies, in minutes, how often the application checks for idle user accounts. The default value is "60" (minutes).

puppet_enterprise::profile::console::rbac_account_expiry_days

An integer that specifies, in days, the duration before an inactive user account expires. The default is undefined. To activate the feature, add a value of "1" or greater.

If a non-superuser hasn't logged into the console during this specified period, their user status updates to revoked. After creating an account, if a non-superuser hasn't logged in to the console during the specified period, their user status updates to revoked.

Note: If you do not specify the `rbac_account_expiry_days` parameter, the `rbac_account_expiry_check_minutes` parameter is ignored.

Configuring proxies

You can work around limited internet access by configuring proxies at various points in your infrastructure, depending on your connectivity limitations.

The examples provided here assume an unauthenticated proxy running at `proxy.example.vlan` on port 8080.

Downloading agent installation packages through a proxy

If your primary server doesn't have internet access, it can't download agent installation packages. If you want to use package management to install agents, set up a proxy and specify its connection details so that `pe_repo` can access agent tarballs.

In the `pe_repo` class of the **PE Master** node group, specify values for `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` settings.

If you want to specify these settings in `pe.conf`, add the following to your `pe.conf` file with your desired parameters:

```
"pe_repo::http_proxy_host": "proxy.example.vlan",
"pe_repo::http_proxy_port": 8080
```

Tip: To test proxy connections to `pe_repo`, run:

```
proxy_uri='http://proxy.example.vlan:8080'
uri='https://pm.puppetlabs.com'

curl --proxy "$proxy_uri" --head "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Setting a proxy for agent traffic

General proxy settings in `puppet.conf` manage HTTP connections that are directly initiated by the agent.

To configure agents to communicate through a proxy using `pe.conf`, specify values for the `http_proxy_host` and `http_proxy_port` settings in `/etc/puppetlabs/puppet/puppet.conf`.

```
http_proxy_host = proxy.example.vlan
http_proxy_port = 8080
```

For more information about HTTP proxy host options, see the Puppet [configuration reference](#).

Setting a proxy for Code Manager traffic

Code Manager has its own set of proxy configuration options which you can use to set a proxy for connections to the Git server or the Forge. These settings are unaffected by the proxy settings in `puppet.conf`, because Code Manager is run by Puppet Server.

Note: To set a proxy for Code Manager connections, you must use an HTTP URL for your `r10k` remote and for all Puppetfile module entries.

Use a proxy for all HTTP connections, including both Git and the Forge, when configuring Code Manager.

To configure Code Manager to use a proxy using Hiera, add the following code to your default `.yaml` and specify your proxy name. For example:

```
puppet_enterprise::profile::master::r10k_proxy: "http://
proxy.example.vlan:8080"
```

Tip: To test proxy connections to Git or the Forge, run one of these commands:

```
proxy_uri='http://proxy.example.vlan:8080'
uri='https://github.com'

curl --proxy "$proxy_uri" --head "$uri"
```

```
proxy_uri='http://proxy.example.vlan:8080'
uri='https://forgeapi.puppet.com'

curl --proxy "$proxy_uri" --head "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

For detailed information about configuring proxies for Code Manager traffic, see the Code Manager documentation.

Related information

[Configuring proxies](#) on page 634

To configure proxy servers, use the proxy setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

Configuring the console

After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.

Configure the PE console and console-services

Configure the behavior of the console and console-services, as needed.

Note: Use the `Hiera` or `pe.conf` method to configure non-profile classes, such as `puppet_enterprise::api_port` and `puppet_enterprise::console_services::no_longer_reporting_cutoff`.

To configure settings in the console:

1. Click **Node groups**, and select the node group that contains the class you want to work with.
2. On the **Classes** tab, find the class you want to work with, select the **Parameter name** from the list and edit its value.
3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Related information

[Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Console and console-services parameters

Use these parameters to customize the behavior of the console and console-services. Parameters that begin with `puppet_enterprise::profile` can be modified from the console itself. See the configuration methods documents for more information on how to change parameters in the console or Hiera.

`puppet_enterprise::profile::console::classifier_synchronization_period`

Integer representing, in seconds, the classifier synchronization period, which controls how long it takes the node classifier to retrieve classes from the primary server.

Default: "600" (seconds).

`puppet_enterprise::profile::console::rbac_failed_attempts_lockout`

Integer specifying how many failed login attempts are allowed on an account before that account is revoked.

Default: "10" (attempts).

`puppet_enterprise::profile::console::rbac_password_reset_expiration`

Integer representing, in hours, how long a user's generated token is valid for. An administrator generates this token for a user so that they can reset their password.

Default: "24" (hours).

`puppet_enterprise::profile::console::rbac_session_timeout`

Integer representing, in minutes, how long a user's session can last. The session length is the same for node classification, RBAC, and the console.

Default: "60" (minutes).

`puppet_enterprise::profile::console::session_maximum_lifetime`

Integer representing the maximum allowable period that a console session can be valid. To not expire before the maximum token lifetime, set to '0'.

Supported units are "s" (seconds), "m" (minutes), "h" (hours), "d" (days), "y" (years). Units are specified as a single letter following an integer, for example "1d"(1 day). If no units are specified, the integer is treated as seconds.

puppet_enterprise::profile::console::console_ssl_listen_port

Integer representing the port that the console is available on.

Default: [443]

puppet_enterprise::profile::console::ssl_listen_address

Nginx listen address for the console.

Default: "0.0.0.0"

puppet_enterprise::profile::console::classifier_prune_threshold

Integer representing the number of days to wait before pruning the size of the classifier database. If you set the value to "0", the node classifier service is never pruned.

puppet_enterprise::profile::console::classifier_node_check_in_storage

"true" to store an explanation of how nodes match each group they're classified into, or "false".

Default: "false"

puppet_enterprise::profile::console::display_local_time

"true" to display timestamps in local time, with hover text showing UTC time, or "false" to show timestamps in UTC time.

Default: "false"

Modify these configuration parameters in `Hiera` or `pe.conf`, not the console:

puppet_enterprise::api_port

SSL port that the node classifier is served on.

Default: [4433]

puppet_enterprise::console_services::no_longer_reporting_cutoff

Length of time, in seconds, before a node is considered unresponsive.

Default: 3600 (seconds)

console_admin_password

The password to log into the console, for example "myconsolepassword".

Default: Specified during installation.

Manage the HTTPS redirect

By default, the console redirects to HTTPS when you attempt to connect over HTTP. You can customize the redirect target URL or disable redirection.

Customize the HTTPS redirect target URL

By default, the redirect target URL is the same as the FQDN of your primary server, but you can customize this redirect URL.

To change the target URL in the console:

1. Click **Node groups**, and select the **PE Infrastructure** node group.
2. On the **Classes** tab, find the `puppet_enterprise::profile::console::proxy::http_redirect` class, select the `server_name` parameter from the list, and change its value to the desired server.
3. Click **Add parameter** and commit changes.
4. On the nodes hosting the primary server and console, run Puppet.

Disable the HTTPS redirect

The pe-nginx webserver listens on port 80 by default. If you need to run your own service on port 80, you can disable the HTTPS redirect.

1. Add the following to your default `.yaml` file with the parameter set to `false`.

```
puppet_enterprise::profile::console::proxy::http_redirect::enable_http_redirect:
  false
```

2. To compile changes, run `puppet agent -t` on the primary server.

Enable data editing in the console

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

1. On your primary server, edit `/etc/puppetlabs/puppet/hiera.yaml` to add:

```
hierarchy:
- name: "Classifier Configuration Data"
  data_hash: classifier_data
```

Place any additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same hierarchy key, preferably below the `Classifier Configuration Data` entry.

Note: If you enable data editing in the console, add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

2. If your environment is configured for disaster recovery, update `hiera.yaml` on your replica.

Add custom PQL queries to the console

Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary sever, as root, copy the `custom_pql_queries.json.example` file and remove the `example` suffix.

```
cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

2. Edit the file contents to include your own PQL queries or remove any existing queries.
3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop down options when running jobs.

Configuring orchestration

After installing PE, you can change some default settings to further configure the orchestrator and pe-orchestration-services.

Configure the orchestrator and pe-orchestration-services

There are several optional parameters you can add to configure the behavior of the orchestrator and pe-orchestration-services. Because they are profile classes, you can change these in the console in the PE Orchestrator group.

puppet_enterprise::profile::orchestrator::task_concurrency

Integer representing the number of simultaneous task or plan actions that can run at the same time in the orchestrator. All task and plan actions are limited by this concurrency limit regardless of transport type (WinRM, SSH, PCP).

Default: "250" (actions)

puppet_enterprise::profile::bolt_server::concurrency

An integer that determines the maximum number of simultaneous task or plan requests orchestrator can make to bolt-server. Only task or plan executions on nodes with SSH or WinRM transport methods are limited by this setting because only they require requests to bolt-server.

Default: "100" (requests)



CAUTION: Do not set a concurrency limit that is higher than the bolt-server limit. This can cause timeouts that lead to failed task runs.

puppet_enterprise::profile::agent::pxp_enabled

Disable or enable the PXP service by setting it to `true` or `false`. If you disable this setting you can't use the orchestrator or the **Run Puppet** button in the console.

Default: `true`

puppet_enterprise::profile::orchestrator::global_concurrent_compiles

An integer that determines how many concurrent compile requests can be outstanding to the primary server, across all orchestrator jobs.

Default: "8" (requests)

puppet_enterprise::profile::orchestrator::job_prune_threshold

Integer that represents the number of days before job reports are removed.

Default: "30" (days)

puppet_enterprise::profile::orchestrator::pcp_timeout

An integer that represents how many seconds must pass while an agent attempts to connect to a PCP broker. If the agent can't connect to the broker in that time frame, the run times out.

Default: "30" (seconds)

puppet_enterprise::profile::orchestrator::run_service

Disable or enable orchestration services. Set to `true` or `false`.

Default: `true`

puppet_enterprise::profile::orchestrator::use_application_services

Enable or disable application management. Set to `true` or `false`.

Default: `false`

puppet_enterprise::profile::orchestrator::allowed_pcp_status_requests

An integer that defines how many times an orchestrator job allows status requests to time out before a job is considered failed. Status requests wait 12 seconds between timeouts, so multiply the value of the `allowed_pcp_status_requests` by 12 to determine how many seconds the orchestrator waits on targets that aren't responding to status requests.

Default: "35" (timeouts)

Related information

[Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Configure the PXP agent

Puppet Execution Protocol (PXP) is a messaging system used to request tasks and communicate task statuses. The PXP agent runs the PXP service and you can configure it using Hiera or the console.

puppet_enterprise::pxp_agent::ping_interval

Controls how frequently (in seconds) PXP agents will ping PCP brokers. If the brokers don't respond, the agents try to reconnect.

Default: 120 (seconds)

puppet_enterprise::pxp_agent::pxp_logfile

A string that represents the path to the PXP agent log file and can be used to debug issues with orchestrator.

Default:

- *nix: /var/log/puppetlabs/pxp-agent/pxp-agent.log
- Windows: C:\Program Data\PuppetLabs\pxp-agent\var\log\pxp-agent.log

puppet_enterprise::pxp_agent::spool_dir_purge_ttl

The amount of time to keep records of old Puppet or task runs on agents. You can declare time in minutes (30m), hours (2h), and days (14d).

Default: 14d

puppet_enterprise::pxp_agent::task_cache_dir_purge_ttl

Controls how long tasks are cached after use. You can declare time in minutes (30m), hours (2h), and days (14d).

Default: 14d

puppet_enterprise::pxp_agent::broker_proxy

Sets a proxy URI used to connect to the pcg-broker to listen for task and Puppet runs.

puppet_enterprise::pxp_agent::master_proxy

Sets a proxy URI used to connect to the primary server to download task implementations.

puppet_enterprise::pcp_max_message_size_mb

Sets the message size, in mb, for pcg_broker, pxp_agent, and the orchestrator. The maximum message size cannot be higher than the default of 64mb, so you can only reduce it.

Default: 64 (mb)

Note: We do not recommend changing the `pcp_max_message_size_mb` parameter if you send or receive large payloads because it might cause errors for large task and plan run parameters and output.

Correct ARP table overflow

In larger deployments that use the PCP broker, you might encounter ARP table overflows and need to adjust some system settings.

Overflows occur when the ARP table—a local cache of IP address to MAC address resolutions—fills and starts evicting old entries. When frequently used entries are evicted, network traffic will increase to restore them, increasing network latency and CPU load on the broker.

A typical log message looks like:

```
[root@s1 peadmin]# tail -f /var/log/messages
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
Aug 10 22:42:36 s1 kernel: Neighbour table overflow.
```

To work around this issue:

Increase sysctl settings related to ARP tables.

For example, the following settings are appropriate for networks hosting up to 2000 agents:

```
# Set max table size
net.ipv6.neigh.default.gc_thresh3=4096
net.ipv4.neigh.default.gc_thresh3=4096
# Start aggressively clearing the table at this threshold
net.ipv6.neigh.default.gc_thresh2=2048
net.ipv4.neigh.default.gc_thresh2=2048
# Don't clear any entries until this threshold
net.ipv6.neigh.default.gc_thresh1=1024
net.ipv4.neigh.default.gc_thresh1=1024
```

Configuring ulimit

As your infrastructure grows and you bring more agents under management, you might need to increase the number allowed file handles per client.

Configure ulimit for PE services

The various services in PE require up to one file handle per connected client. On most operating system configurations, the defaults are not adequate for more than a couple hundred clients. To support more clients, you need to increase the number of allowed file handles.

You can increase the limits for the following services:

- pe-orchestration-services
- pe-puppetdb
- pe-console-services
- pe-puppetserver
- pe-puppet

The location and method for configuring ulimit depends on your agent's platform. You might use systemd, upstart, or some other init system.

In the following instructions, replace <PE SERVICE> with the specific service you're editing. The examples show setting a limit of 32768, which you can also change according to what you need.

Configure ulimit using systemd

With systemd, the number of open file handles allowed is controlled by a setting in the service file at `/usr/lib/systemd/system/<PE SERVICE>.service`.

1. To increase the limit, run the following commands, setting the `LimitNOFILE` value to the new number:

```
mkdir /etc/systemd/system/<PE SERVICE>.service.d
echo "[Service]
LimitNOFILE=32768" > /etc/systemd/system/<PE SERVICE>.service.d/
limits.conf
systemctl daemon-reload
```

2. Confirm the change by running: `systemctl show <PE SERVICE> | grep LimitNOFILE`

Configure ulimit using upstart

For Ubuntu and Red Hat systems, the number of open file handles allowed for is controlled by settings in service files.

The service files are:

- Ubuntu: `/etc/default/<PE SERVICE>`
- Red Hat: `/etc/sysconfig/<PE SERVICE>`

For both Ubuntu and Red Hat, set the last line of the file as follows:

```
ulimit -n 32768
```

This sets the number of open files allowed at 32,678.

Configure ulimit on other init systems

The ulimit controls the number of processes and file handles that the PE service user can open and process.

To increase the ulimit for a PE service user:

Edit `/etc/security/limits.conf` so that it contains the following lines:

```
<PE SERVICE USER> soft nofile 32768
<PE SERVICE USER> hard nofile 32768
```

Writing configuration files

Puppet supports two formats for configuration files that configure settings: valid JSON and Human-Optimized Config Object Notation (HOCON), a JSON superset.

For more information about HOCON itself, see the [HOCON documentation](#).

Configuration file syntax

Refer to these examples when you're writing configuration files to identify correct JSON or HOCON syntax.

Brackets

In HOCON, you can omit the brackets (`{ }`) around a root object.

JSON example	HOCON example
<pre>{ "authorization": { "version": 1 } }</pre>	<pre>"authorization": { "version": 1 }</pre>

Quotes

In HOCON, double quotes around key and value strings are optional in most cases. However, double quotes are required if the string contains the characters `*`, `^`, `+`, `:`, or `=`.

JSON example	HOCON example
<pre>"authorization": { "version": 1 }</pre>	<pre>authorization: { version: 1 }</pre>

Commas

When writing a map or array in HOCON, you can use a new line instead of a comma.

	JSON example	HOCON example
Map	<pre>rbac: { password-reset- expiration: 24, session-timeout: 60, failed-attempts- lockout: 10, }</pre>	<pre>rbac: { password-reset- expiration: 24 session-timeout: 60 failed-attempts- lockout: 10 }</pre>
Array	<pre>http-client: { ssl-protocols: [TLSv1, TLSv1.1, TLSv1.2] }</pre>	<pre>http-client: { ssl-protocols: [TLSv1 TLSv1.1 TLSv1.2] }</pre>

Comments

Add comments using either // or #. Inline comments are supported.

HOCON example
<pre>authorization: { version: 1 rules: [{ # Allow nodes to retrieve their own catalog match-request: { path: "^/puppet/v3/catalog/([^/]+)\$" type: regex method: [get, post] } }] }</pre>

Analytics data collection

Some components automatically collect data about how you use Puppet Enterprise. If you want to opt out of providing this data, you can do so, either during or after installing.

What data does Puppet Enterprise collect?

Puppet Enterprise (PE) collects the following data when Puppet Server starts or restarts, and again every 24 hours.

License, version, primary server, and agent information:

- License UUID
- Number of licensed nodes
- Product name
- PE version

- Primary server's operating system
- Primary server's public IP address
- Whether the primary server is running on Microsoft Azure
- The hypervisor the primary server is running on, if applicable
- Number of nodes in deployment
- Agent operating systems
- Number of agents running each operating system
- Agent versions
- Number of agents running each version of Puppet agent
- All-in-One (AIO) puppet-agent package versions
- Number of agents running on Microsoft Azure or Google Cloud Platform, if applicable
- Number of configured disaster recovery replicas, if applicable

Puppet Enterprise feature use information:

- Number of node groups in use
- Number of nodes used in orchestrator jobs after last orchestrator restart
- Mean nodes per orchestrator job
- Maximum nodes per orchestrator job
- Minimum nodes per orchestrator job
- Total orchestrator jobs created after last orchestrator restart
- Number of non-default user roles in use
- Type of certificate autosigning in use
- Number of nodes in the job that were run over Puppet Communications Protocol
- Number of nodes in the job that were run over SSH
- Number of nodes in the job that were run over WinRM
- Number of nodes patched per task run
- Type of operating system on nodes patched in a task run
- Number of patches applied to each node per task run
- Number of patches completed per task run
- Number of nodes with the `pe_patch` module
- Number of nodes with the `pe_patch` module that require patching
- List of Puppet task jobs
- List of Puppet deploy jobs
- List of Puppet task jobs run by plans
- List of file upload jobs run by plans
- List of script jobs run by plans
- List of command jobs run by plans
- List of wait jobs run by plans
- How nodes were selected for the job
- Whether the job was started by the PE admin account
- Number of nodes in the job
- Length of description applied to the job
- Length of time the job ran
- User agent used to start the job (to distinguish between the console, command line, and API)
- UUID used to correlate multiple jobs run by the same user
- Time the task job was run
- How nodes were selected for the job
- Whether the job was started by the PE admin account
- Number of nodes in the job
- Length of description applied to the job

- Whether the job was asked to override agent-configured no-operation (no-op) mode
- Whether app-management was enabled in the orchestrator for this job
- Time the deploy job was run
- Type of version control system webhook
- Whether the request was to deploy all environments
- Whether code-manager will wait for all deploys to finish or error to return a response
- Whether the deploy is a dry-run
- List of environments requested to deploy
- List of deploy requests
- Total time elapsed for all deploys to finish or error
- List of total wait times for deploys specifying `--wait` option
- Name of environment deployed
- Time needed for r10k to run
- Time spent committing to file sync
- Time elapsed for all environment hooks to run
- List of individual environment deploys
- Puppet classes applied from publicly available modules, with node counts per class

Backup and Restore information:

- Whether user used `--force` option when running restore
- Scope of restore
- Time in seconds for various restore functions
- Time to check for disk space to restore
- Time to stop PE related services
- Time to restore PE file system components
- Time to migrate PE configuration for new server
- Time to configure PE on newly restored primary server
- Time to update PE classification for new server
- Time to deactivate the old primary server node
- Time to restore the pe-orchestrator database
- Time to restore the pe-rbac database
- Time to restore the pe-classifier database
- Time to restore the pe-activity database
- Time to restore the pe-puppetdb database
- Total time to restore
- List of puppet backup restore jobs
- Whether user used `--force` option when running `puppet-backup create`
- Whether user used `--dir` option when running `puppet-backup create`
- Whether user used `--name` option when running `puppet-backup create`
- Scope of backup
- Time in seconds for various back up functions
- Time needed to estimate backup size, disk space needed, and disk space available
- Time to create file system backup
- Time to back up the pe-orchestrator database
- Time to back up the pe-rbac database
- Time to back up the pe-classifier database
- Time to back up the pe-activity database
- Time to back up the pe-puppetdb database
- Time to compress archive file to backup directory
- Time to back up PE related classification

- Total time to back up
- List of puppet-backup create jobs

Puppet Server performance information:

- Total number of JRuby instances
- Maximum number of active JRuby instances
- Maximum number of requests per JRuby instance
- Average number of instances not in use over the process's lifetime
- Average wait time to lock the JRuby pool
- Average time the JRuby pool held a lock
- Average time an instance spent handling requests
- Average time spent waiting to reserve an instance from the JRuby pool
- Number of requests that timed out while waiting for a JRuby instance
- Amount of memory the JVM starts with
- Maximum amount of memory the JVM is allowed to request from the operating system

Installer information:

- Installation method (express, text, web, repair)
- Current version, if upgrading
- Target version
- Success or failure, and limited information on the type of failure, if an

If PE is installed using an Amazon Web Services Marketplace Image:

- The marketplace name
- Marketplace image billing mode (bring your own license or pay as you go)

While in use, the console collects the following information:

- Pageviews
- Link and button clicks
- Page load time
- User language
- Screen resolution
- Viewport size
- Anonymized IP address

The console *does not* collect user inputs such as node or group names, user names, rules, parameters, or variables

The collected data is tied to a unique, anonymized identifier for each primary server and your site as a whole. No personally identifiable information is collected, and the data we collect is never used or shared outside Puppet, Inc.

How does sharing this data benefit you?

We use the data to identify organizations that could be affected by a security issue, alert them to the issue, and provide them with steps to take or fixes to download. In addition, the data helps us understand how people use the product, which helps us improve the product to meet your needs.

How does Puppet use the collected data?

The data we collect is one of many methods we use for learning about our customers. For example, knowing how many nodes you manage helps us develop more realistic product testing. And learning which operating systems are the most and the least used helps us decide where to prioritize new functionality. By collecting data, we begin to understand you as a customer.

Opt out during the installation process

To opt out of data collection during installation, you can set the `DISABLE_ANALYTICS` environment variable when you run the installer script.

Setting the `DISABLE_ANALYTICS` environment variable during installation sets

`puppet_enterprise::send_analytics_data: false` in `pe.conf`, opting you out of data collection.

Follow the instructions for your chosen installation method, adding `DISABLE_ANALYTICS=1` when you call the installer script, for example:

```
sudo DISABLE_ANALYTICS=1 ./puppet-enterprise-installer
```

Opt out after installing

If you've already installed PE and want to disable data collection, follow these steps.

1. In the console, click **Node groups**, and then click **PE Infrastructure**.
2. On the **Classes** tab, in the `puppet_enterprise` class, add `send_analytics_data` as a parameter and set the **Value** to `false`.
3. On the **Inventory** page, select your primary server and click **Run Puppet**.
4. Select your console node and click **Run Puppet**.

After Puppet runs to enforce the changes on the master and console nodes, you have opted out of data collection.

Static catalogs

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing `source` attributes pointing to `puppet:///` locations on a node.

The metadata in a static catalog can refer to a specific version of the file (not just the latest version), and can confirm that the agent is applying the appropriate version of the file resource for the catalog. Because the metadata is provided in the catalog, agents make fewer requests to the primary server.

See the open source Puppet documentation more information about [Resources](#), [File types](#), and [Catalog compilation](#).

Enabling static catalogs

When a primary server produces a non-static catalog, the catalog doesn't specify the version of file resources. When the agent applies the catalog, it always retrieves the latest version of that file resource, or uses a previously retrieved version if it matches the latest version's contents.

This potential problem affects file resources that use the `source` attribute. File resources that use the `content` attribute are not affected, and their behavior does not change in static catalogs.

When a manifest depends on a file whose contents change more frequently than the agent receives new catalogs, a node might apply a version of the referenced file that doesn't match the instructions in the catalog. In Puppet Enterprise (PE), such situations are particularly likely if you've configured your agents to run off cached catalogs for participation in application orchestration services.

Consequently, the agent's Puppet runs might produce different results each time the agent applies the same catalog. This often causes problems because Puppet generally expects a catalog to produce the same results each time it's applied, regardless of any code or file content updates on the primary server.

Additionally, each time an agent applies a normal cached catalog that contains file resources sourced from `puppet:///` locations, the agent requests file metadata from the primary server each time the catalog's applied, even though nothing's changed in the cached catalog. This causes the primary server to perform unnecessary resource-intensive checksum calculations for each file resource.

Static catalogs avoid these problems by including metadata that refers to a specific version of the resource's file. This prevents a newer version from being incorrectly applied, and avoids having the agent regenerate the metadata on each Puppet run. The metadata is delivered in the form of a unique hash maintained, by default, by the file sync service.

We call this type of catalog "static" because it contains all of the information that an agent needs to determine whether the node's configuration matches the instructions and state of file resources at the static point in time when the catalog was generated.

Differences in catalog behavior

Without static catalogs enabled:

- The agent sends facts to the primary server and requests a catalog.
- The primary server compiles and returns the agent's catalog.
- The agent applies the catalog by checking each resource the catalog describes. If it finds any resources that are not in the desired state, it makes the necessary changes.

With static catalogs enabled:

- The agent sends facts to the primary server and requests a catalog.
- The primary server compiles and returns the agent's catalog, including metadata that specifies the desired state of the node's file resources.
- The agent applies the catalog by checking each resource the catalog describes. If the agent finds any resources that are not in the desired state, it makes the necessary changes based on the state of the file resources at the static point in time when the catalog was generated.
- If you change code on the primary server, file contents are not updated until the agent requests a new catalog with new file metadata.

Enabling file sync

In PE, static catalogs are disabled across all environments for new installations. To use static catalogs in PE, you must enable file sync. After file sync is enabled, Puppet Server automatically creates static catalogs containing file metadata for eligible resources, and agents running Puppet 1.4.0 or newer can take advantage of the catalogs' new features.

If you do not enable file sync and Code Manager, you can still use static catalogs, but you need to create some custom scripts and set a few parameters in the console.

Enforcing change with static catalogs

When you are ready to deploy new Puppet code and deliver new static catalogs, you don't need to wait for agents to check in. Use the Puppet orchestrator to enforce change and deliver new catalogs across your PE infrastructure, on a per-environment basis.

When aren't static catalogs applied?

In the following scenarios, either agents won't apply static catalogs, or catalogs won't include metadata for file resources.

- Static catalogs are globally disabled.
- Code Manager and file sync are disabled, and `code_id` and `code_content` aren't configured.
- Your agents aren't running PE 2016.1 or later (Puppet agent version 1.4.0 or later).

Additionally, Puppet won't include metadata for a file resource if it:

- Uses the `content` attribute instead of the `source` attribute.
- Uses the `source` attribute with a non-Puppet scheme (for example `source => 'http://host:port/path/to/file'`).
- Uses the `source` attribute without the built-in modules mount point.

- Uses the `source` attribute, but the file on the primary server is not in `/etc/puppetlabs/code/environments/<environment>/**/*.files/**`. For example, module files are typically in `/etc/puppetlabs/code/environments/<environment>/modules/<module_name>/files/**`.

Agents continue to process the catalogs in these scenarios, but without the benefits of inlined file metadata or file resource versions.

Related information

[Enabling or disabling file sync](#) on page 676

File sync is normally enabled or disabled automatically along with Code Manager.

[How Puppet orchestrator works](#) on page 461

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

Disabling static catalogs globally with Hiera

You can turn off all use of static catalogs with a Hiera setting.

To disable static catalogs using Hiera:

1. Add the following code to your default `.yaml` file and set the parameter to `false`:

```
puppet_enterprise::master::static_catalogs: false
```

2. To compile changes, run `puppet agent -t`

Using static catalogs without file sync

To use static catalogs without enabling file sync, you must set the `code_id` and `code_content` parameters in Puppet, and then configure the `code_id_command`, `code_content_command`, and `file_sync_enabled` parameters in the console.

1. Set `code_id` and `code_content` by following the instructions in the [open source Puppet static catalogs documentation](#). Then return to this page to set the remaining parameters.
2. In the console, click **Node groups**, and in the **PE Infrastructure** node group, select the **PE Master** node group.
3. On the **Classes** tab, locate the `puppet_enterprise::profile::master` class, and select **file_sync_enabled** from its **Parameter** list.
4. In the **Value** field, enter `false`, and click **Add parameter**.
5. Select the **code_id_command** parameter, and in the **Value** field, enter the absolute path to the `code_id` script, and click **Add parameter**.
6. Select the **code_content_command** parameter, and in the **Value** field, add the absolute to the `code_content` script, and click **Add parameter**.
7. Commit changes.
8. Run Puppet on the primary server.

Related information

[Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Configuring disaster recovery

Enabling disaster recovery for Puppet Enterprise ensures that your systems can fail over to a replica of your primary server if infrastructure components become unreachable.

- [Disaster recovery](#) on page 185

Disaster recovery creates a replica of your primary server.

- [Configure disaster recovery](#) on page 189

To configure disaster recovery, you must provision a replica to serve as backup during failovers. If your primary server is permanently disabled, you can then promote a replica.

Disaster recovery

Disaster recovery creates a replica of your primary server.

You can have only one replica at a time, and you can add disaster recovery to an installation with or without compilers. Disaster recovery isn't supported with standalone PE-PostgreSQL or FIPS-compliant installations.

There are two main advantages to enabling disaster recovery:

- If your primary server fails, the replica takes over, continuing to perform critical operations.
- If your primary server can't be repaired, you can promote the replica to primary server. Promotion establishes the replica as the new, permanent primary server.

Related information

[What happens during failovers](#) on page 186

Failover occurs when the replica takes over services usually performed by the primary server.

Disaster recovery architecture

The replica is not an exact copy of the primary server. Rather, the replica duplicates specific infrastructure components and services. Hieradata and other custom configurations are not replicated.

Replication can be *read-write*, meaning that data can be written to the service or component on either the primary server or the replica, and the data is synced to both nodes. Alternatively, replication can be *read-only*, where data is written only to the primary server and synced to the replica. Some components and services, like Puppet Server and the console service UI, are not replicated because they contain no native data.

Some components and services are activated immediately when you enable a replica; others aren't active until you promote a replica. After you provision and enable a replica, it serves as a compiler, redirecting PuppetDB and cert requests to the primary server.

Component or service	Type of replication	Activated when replica is...
Puppet Server	none	enabled
File sync client	read-only	enabled
PuppetDB	read-write	enabled
Certificate authority	read-only	promoted
RBAC service	read-only	enabled
Node classifier service	read-only	enabled
Activity service	read-only	enabled

Component or service	Type of replication	Activated when replica is...
Orchestration service	read-only	promoted
Console service UI	none	promoted
Agentless Catalog Executor (ACE) service	none	promoted
Bolt service	none	promoted

In a standard installation, when a Puppet run fails over, agents communicate with the replica instead of the primary server. In a large or extra-large installation with compilers, agents communicate with load balancers or compilers, which communicate with the primary server or replica.

Related information

[Configure settings with Hiera](#) on page 161

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

What happens during failovers

Failover occurs when the replica takes over services usually performed by the primary server.

Failover is automatic — you don’t have to take action to activate the replica. With disaster recovery enabled, Puppet runs are directed first to the primary server. If the primary server is either fully or partially unreachable, runs are directed to the replica.

In partial failovers, Puppet runs can use the server, node classifier, or PuppetDB on the replica if those services aren’t reachable on the primary server. For example, if the primary server’s node classifier fails, but its Puppet Server is still running, agent runs use the Puppet Server on the primary server but fail over to the replica’s node classifier.

What works during failovers:

- Scheduled Puppet runs
- Catalog compilation
- Viewing classification data using the node classifier API
- Reporting and queries based on PuppetDB data

What doesn’t work during failovers:

- Deploying new Puppet code
- Editing node classifier data
- Using the console
- Certificate functionality, including provisioning new agents, revoking certificates, or running the `puppet certificate` command
- Most CLI tools
- Application orchestration

System and software requirements for disaster recovery

Your Puppet infrastructure must meet specific requirements in order to configure disaster recovery.

Component	Requirement
Operating system	All supported PE primary server platforms.

Component	Requirement
Software	<ul style="list-style-type: none"> You must use Code Manager so that code is deployed to both the primary server and the replica after you enable a replica. You must use the default PE node classifier so that disaster recovery classification can be applied to nodes. Orchestrator must be enabled so that agents are updated when you provision or enable a replica. Orchestrator is enabled by default.
Replica	<ul style="list-style-type: none"> Must be an agent node that doesn't have a specific function already. You can decommission a node, uninstall all puppet packages, and re-commission the node to be a replica. However, a compiler cannot perform two functions, for example, as a compiler and a replica. Must have the same hardware specifications and capabilities as your primary server.
Firewall	<p>Both the primary server and the replica must comply with these port requirements:</p> <ul style="list-style-type: none"> Firewall configuration for installations with compilers. These requirements apply whether your disaster recovery environment uses a single primary server or compilers. Port 5432 must be open to facilitate database replication by console services.
Node names	<ul style="list-style-type: none"> You must use resolvable domain names when specifying node names for the primary server and replica.
RBAC tokens	<ul style="list-style-type: none"> You must have an admin RBAC token when running <code>puppet infrastructure</code> commands, including <code>provision</code>, <code>enable</code>, and <code>forget</code>. You can generate a token using the <code>puppet-access</code> command. <p>Note: You don't need an RBAC token to promote a replica.</p>

Related information

[Managing and deploying Puppet code](#) on page 614

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

[Configure the orchestrator and pe-orchestration-services](#) on page 173

There are several optional parameters you can add to configure the behavior of the orchestrator and pe-orchestration-services. Because they are profile classes, you can change these in the console in the PE Orchestrator group.

[Firewall configuration for large installations](#) on page 90

These are the port requirements for large installations with compilers.

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Classification changes in disaster recovery installations

When you provision and enable a replica, the system makes a number of classification changes in order to manage disaster recovery without affecting existing configuration.

These preconfigured node groups are added in disaster recovery installations:

Node group	Matching nodes	Inherits from
PE HA Master	primary server	PE Master node group
PE HA Replica	replica	PE Infrastructure node group

These parameters are used to configure disaster recovery installations:

Parameter	Purpose	Node group	Classes	Notes
<code>agent-server-urls</code>	Specifies the list of servers that agents contact, in order.	PE Agent PE Infrastructure Agent	<code>puppet_enterprise::profile::agent</code>	In large installations with compilers, agents must be configured to communicate with the load balancers or compilers. Important: Setting agents to communicate directly with the replica in order to use the replica as a compiler is not a supported configuration.
<code>replication_mode</code>	Sets replication type and direction on primary servers and replicas.	PE Master (none) HA Master (source) HA Replica (replica)	<code>puppet_enterprise::profile::master</code> <code>puppet_enterprise::profile::database</code> <code>puppet_enterprise::profile::console</code>	System parameter. Don't modify.

Parameter	Purpose	Node group	Classes	Notes
ha_enabled_replicas	Tracks replica nodes that are failover ready.	PE Infrastructure	puppet_enterprise	System parameter. Don't modify. Updated when you enable a replica.
pcp_broker_list	Specifies the list of Puppet Communications Protocol brokers that Puppet Execution Protocol agents contact, in order.	PE Agent PE Infrastructure Agent	puppet_enterprise::profile::agent	

Load balancer timeout in disaster recovery installations

Disaster recovery configuration uses timeouts to determine when to fail over to the replica. If the load balancer timeout is shorter than the server and agent timeout, connections from agents might be terminated during failover.

To avoid timeouts, set the timeout option for load balancers to four minutes or longer. This duration allows compilers enough time for required queries to PuppetDB and the node classifier service. You can set the load balancer timeout option using parameters in the haproxy or f5 modules.

Configure disaster recovery

To configure disaster recovery, you must provision a replica to serve as backup during failovers. If your primary server is permanently disabled, you can then promote a replica.

Before you begin

Apply [disaster recovery system and software requirements](#).

Tip: The `puppet infrastructure` commands, which are used to configure and manage disaster recovery, require a valid admin RBAC token and must be run from a root session. Running with elevated privileges via `sudo puppet infrastructure` is not sufficient. Instead, start a root session by running `sudo su -`, and then run the `puppet infrastructure` command. For details about these commands, run `puppet infrastructure help <ACTION>`, for example, `puppet infrastructure help provision`.

Related information

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Provision and enable a replica

Provisioning a replica duplicates specific components and services from the primary server to the replica. Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.

Before you begin

- Ensure you have a valid admin RBAC token.
- Ensure Code Manager is enabled and configured on your primary server.
- Move any tuning parameters that you set for your primary server using the console to Hiera. Using Hiera ensures configuration is applied to both your primary server and replica.
- Back up your classifier hierarchy, because enabling a replica alters classification.

Note: While completing this task, the primary server is unavailable to serve catalog requests. Time completing this task accordingly.

1. Configure infrastructure agents to connect orchestration agents to the primary server.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent > PE Infrastructure Agent** group.
 - b) If you manage your load balancers with agents, on the **Rules** tab, pin load balancers to the group. Pinning load balancers to the **PE Infrastructure Agent** group ensures that they communicate directly with the primary server.
 - c) On the **Classes** tab, find the **puppet_enterprise::profile::agent** class and specify these parameters:

Parameter	Value
manage_puppet_conf	Specify true to ensure that your setting for <code>server_list</code> is configured in the expected location and persists through Puppet runs.
pcp_broker_list	Hostname for your primary server. Hostnames must include port 8142, for example <code>["PRIMARY.EXAMPLE.COM:8142"]</code> .
master_uris server_list	Hostname for your primary server, for example <code>["PRIMARY.EXAMPLE.COM"]</code> . This setting assumes port 8140 unless you specify otherwise with <code>host:port</code> .

- d) Remove any values set for **pcp_broker_ws_uris**.
 - e) Commit changes.
 - f) Run Puppet on all agents classified into the **PE Infrastructure Agent** group.
2. On the primary server, as the root user, run `puppet infrastructure provision replica <REPLICA NODE NAME> --enable`

Note: In installations with compilers, use the `--skip-agent-config` flag with the `--enable` option if you want to:

- Upgrade a replica without needing to run Puppet on all agents.
- Add disaster recovery to an installation without modifying the configuration of existing load balancers.
- Manually configure which load balancer agents communicate with in geo-diverse installations. See [Managing agent communication in geo-diverse installations](#) on page 191.

3. Copy your secret key file from the primary server to the replica. The path to the secret key file is `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`.

Important: If you do not copy your secret key file onto your replica, the replica generates a new secret key when you promote it. The new key prevents you from accessing credentials for your agentless nodes or running tasks and plans on agentless nodes.

4. Optional: Verify that all services running on the primary server are also running on the replica:
 - a) From the primary server, run `puppet infrastructure status --verbose` to verify that the replica is available.
 - b) From any managed node, run `puppet agent -t --noop --server_list=<REPLICA HOSTNAME>`. If the replica is correctly configured, the Puppet run succeeds and shows no changed resources.
5. Optional: Deploy updated configuration to agents by running Puppet, or wait for the next scheduled Puppet run.

If you used the `--skip-agent-config` option, you can skip this step.

Note: If you use the direct Puppet workflow, where agents use cached catalogs, you must manually deploy the new configuration by running `puppet job run --no-enforce-environment --query 'nodes {deactivated is null and expired is null}'`

6. Optional: Perform any tests you feel are necessary to verify that Puppet runs continue to work during failover. For example, to simulate an outage on the primary server:
 - a) Prevent the replica and a test node from contacting the primary server. For example, you might temporarily shut down the primary server or use `iptables` with drop mode.
 - b) Run `puppet agent -t` on the test node. If the replica is correctly configured, the Puppet run succeeds and shows no changed resources. Runs might take longer than normal when in failover mode.
 - c) Reconnect the replica and test node.

Related information

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

[Configure settings with Hieradata](#) on page 161

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

[Back up your infrastructure](#) on page 687

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

[Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

Managing agent communication in geo-diverse installations

Typically, when you enable a replica using `puppet infrastructure enable replica`, the configuration tool automatically sets the same communication parameters for all agents. In *geo-diverse installations*, with load balancers or compilers in multiple locations, you must manually configure agent communication settings so that agents fail over to the appropriate load balancer or compiler.

To skip automatically configuring which Puppet servers and PCP brokers agents communicate with, use the `--skip-agent-config` flag when you provision and enable a replica, for example:

```
puppet infrastructure provision replica example.puppet.com --enable --skip-agent-config
```

To manually configure which load balancer or compiler agents communicate with, use one of these options:

- CSR attributes
 1. For each node, include a CSR attribute that identifies the location of the node, for example `pp_region` or `pp_datacenter`.
 2. Create child groups off of the **PE Agent** node group for each location.
 3. In each child node group, include the `puppet_enterprise::profile::agent` module and set the `server_list` parameter to the appropriate load balancer or compiler hostname.
 4. In each child node group, add a rule that uses the trusted fact created from the CSR attribute.
- Hiera

For each node or group of nodes, create a key/value pair that sets the `puppet_enterprise::profile::agent::server_list` parameter to be used by the **PE Agent** node group.

- Custom method that sets the `server_list` parameter in `puppet.conf`.

Promote a replica

If your primary server can't be restored, you can promote the replica to primary server to establish the replica as the new, permanent primary server.

1. Verify that the primary server is permanently offline.

If the primary server comes back online during promotion, your agents can get confused trying to connect to two active primary servers.

2. On the replica, as the root user, run `puppet infrastructure promote replica`

Promotion can take up to the amount of time it took to install PE initially. Don't make code or classification changes during or after promotion.

3. When promotion is complete, update any systems or settings that refer to the old primary server, such as PE client tool configurations, Code Manager hooks, and CNAME records.

4. Deploy updated configuration to nodes by running Puppet or waiting for the next scheduled run.

Note: If you use the direct Puppet workflow, where agents use cached catalogs, you must manually deploy the new configuration by running `puppet job run --no-enforce-environment --query 'nodes {deactivated is null and expired is null}'`

5. If you have a SAML identity provider (IdP) configured for single sign-on access in PE, specify your replica's new URLs and certificate in your IdP's configuration.

View the replica's URLs and certificate in the console on the **Access control** page, on the **SSO** tab, under **Show configuration information**. Because your SAML IdP isn't connected to your replica yet, you'll need to log into the console using a local PE or LDAP account to get the URLs and certificate.

6. Optional: Provision a new replica in order to maintain disaster recovery.

Note: Agent configuration must be updated before provisioning a new replica. If you re-use your old primary server's node name for the new replica, agents with outdated configuration might use the new replica as a primary server before it's fully provisioned.

Related information

[Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

[Connect to a SAML identity provider](#) on page 212

Use the console to set up SSO or MFA with your SAML identity provider.

Enable a new replica using a failed primary server

After promoting a replica, you can use your old primary server as a new replica, effectively swapping the roles of your failed primary server and promoted replica.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see [Bolt OpenSSH configuration options](#).

To view all available parameters, use the `--help` flag. The logs for all `puppet infrastructure run` Bolt plans are located at `/var/log/puppetlabs/installer/bolt_info.log`.

You must be able to reach the failed primary server via SSH from the current primary server.

On your promoted replica, as the root user, run `puppet infrastructure run enable_ha_failover`, specifying these parameters:

- `host` — Hostname of the failed primary server. This node becomes your new replica.

- `topology` — Architecture used in your environment, either `mono` (standard) or `mono-with-compile` (large).
- `replication_timeout_secs` — Optional. The number of seconds allowed to complete provisioning and enabling of the new replica before the command fails.
- `tmpdir` — Optional. Path to a directory to use for uploading and executing temporary files.

For example:

```
puppet infrastructure run enable_ha_failover host=<FAILED_PRIMARY_HOSTNAME>
topology=mono
```

The failed primary server is repurposed as a new replica.

Forget a replica

Forgetting a replica cleans up classification and database state, preventing degraded performance over time.

Before you begin

Ensure you have a valid admin RBAC token.

Run the `forget` command whenever a replica node is destroyed, even if you plan to replace it with a replica with the same name.

1. On the primary server, as the root user, run `puppet infrastructure forget <REPLICA NODE NAME>`
2. Delete your secret key file from the replica because leaving sensitive information on a replica poses a security risk. The path to the secret key file is `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`

The replica is decommissioned, the node is purged as an agent, secret key information is deleted, and a Puppet run is completed on the primary server.

Related information

[Generate a token using puppet-access](#) on page 220

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Reinitialize a replica

If you encounter certain errors on your replica after provisioning, you can reinitialize the replica. Reinitializing destroys and re-creates replica databases, as specified.

Before you begin

Your primary server must be fully functional and the replica must be able to communicate with the primary server.



CAUTION: If you reinitialize a functional replica that you already enabled, the replica is unavailable to serve as backup in a failover during reinitialization.

Reinitialization is not intended to fix slow queries or intermittent failures. Reinitialize your replica only if it's inoperational or you see replication errors.

1. On the replica, as the root user, reinitialize databases as needed:
 - All databases: `puppet infrastructure reinitialize replica`
 - Specific databases: `puppet infrastructure reinitialize replica --db <DATABASE>` where `<DATABASE>` is `pe-activity`, `pe-classifier`, `pe-orchestrator`, or `pe-rbac`.
2. Follow prompts to complete the reinitialization.

Accessing the console

The console is the web interface for Puppet Enterprise.

Use the console to:

- Manage node requests to join the Puppet deployment.
- Assign Puppet classes to nodes and groups.
- Run Puppet on specific groups of nodes.
- View reports and activity graphs.
- Browse and compare resources on your nodes.
- View package and inventory data.
- Manage console users and their access privileges.

Reaching the console

The console is served as a website over SSL, on whichever port you chose when installing the console component.

Let's say your console server is `console.domain.com`. If you chose to use the default port (443), you can omit the port from the URL and reach the console by navigating to `https://console.domain.com`.

If you chose to use port 8443, you reach the console at `https://console.domain.com:8443`.

Remember: Always use the `https` protocol handler. You cannot reach the console over plain `http`.

Accepting the console's certificate

The console uses an SSL certificate created by your own local Puppet certificate authority. Because this authority is specific to your site, web browsers won't know it or trust it, and you must add a security exception in order to access the console.

Adding a security exception for the console is safe to do. Your web browser warns you that the console's identity hasn't been verified by one of the external authorities it knows of, but that doesn't mean it's untrustworthy. Because you or another administrator at your site is in full control of which certificates the Puppet certificate authority signs, the authority verifying the site is *you*.

When your browser warns you that the certificate authority is invalid or unknown:

- In Chrome, click **Advanced**, then **Proceed to <CONSOLE ADDRESS>**.
- In Firefox, click **Advanced**, then **Add exception**.
- In Internet Explorer or Microsoft Edge, click **Continue to this website (not recommended)**.
- In Safari, click **Continue**.

Logging in

Accessing the console requires a username and password.

If you are an administrator setting up the console or accessing it for the first time, use the username and password you chose when you installed the console. Otherwise, get credentials from your site's administrator.

Because the console is the main point of control for your infrastructure, it is a good idea to prohibit your browser from storing the login credentials.

Generate a user password reset token

When users forget passwords or lock themselves out of the console by attempting to log in with incorrect credentials too many times, you need to generate a password reset token.

1. In the console, on the **Access control** page, click the **Users** tab.
2. Click the name of the user who needs a password reset token.
3. Click **Generate password reset**. Copy the link provided in the message and send it to the user.

Reset the console administrator password

If you're unable to log in to the console as admin, you can change the password from the command line of the node running console services.

Log in as root to the node running console services (usually your primary server) and reset the console admin password:

```
puppet infrastructure console_password --password=<MY_PASSWORD>
```

Troubleshooting login to the PE admin account

If your directory contains multiple users with a login name of "admin," the PE admin account is unable to log in.

If you are locked out of PE as the admin user and there are no other users with administrator access who you can ask to reset the access control settings in the console, SSH into the box and use curl commands to reset the directory service settings.

For a box named centos7 the curl call looks like this:

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/rbac-api/v1/ds"
data='{}'

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
--request PUT "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Managing access

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

By using permissions, you give the appropriate level of access and agency to each user. For example, you can grant users:

- The permission to grant password reset tokens to other users who have forgotten their passwords
- The permission to edit a local user's metadata
- The permission to deploy Puppet code to specific environments
- The permission to edit class parameters in a node group

You can do access control tasks in the console or using the RBAC API.

- [User permissions and user roles](#) on page 196

The "role" in role-based access control refers to a system of user roles, which are assigned to user groups and their users. Those roles contain permissions, which define what a user can or can't do within PE.

- [Creating and managing local users and user roles](#) on page 202

Puppet Enterprise's role-based access control (RBAC) enables you to manage users—what they can create, edit, or view, and what they can't—in an organized, high-level way that is vastly more efficient than managing user permissions on a per-user basis. User roles are sets of permissions you can apply to multiple users. You can't assign permissions to single users in PE, only to user roles.

- [Connecting LDAP external directory services to PE](#) on page 204

Puppet Enterprise connects to external Lightweight Directory Access Protocol (LDAP) directory services through its role-based access control (RBAC) service. Because PE integrates with cloud LDAP service providers such as Okta, you can use existing users and user groups that have been set up in your external directory service.

- [Working with user groups from a LDAP external directory](#) on page 209

You don't explicitly add remote users to PE. Instead, after the external directory service has been successfully connected, remote users must log into PE, which creates their user record.

- [Connect a SAML identity provider to PE](#) on page 210

Connect to a Security Assertion Markup Language (SAML) identity provider to log in to PE with single sign-on (SSO). SSO authentication securely centralizes sensitive data and reduces the number of login credentials users have to remember and store. Depending on your identity provider, you can also use this workflow to connect and configure multifactor authentication (MFA) in PE.

- [Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

- [RBAC API v1](#) on page 223

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

- [RBAC API v2](#) on page 252

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

- [Activity service API](#) on page 257

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

User permissions and user roles

The "role" in role-based access control refers to a system of user roles, which are assigned to user groups and their users. Those roles contain permissions, which define what a user can or can't do within PE.

When you add new users to PE, they can't actually do anything until they're associated with a user role, either explicitly through role assignment or implicitly through group membership and role inheritance. When a user is added to a role, they receive all the permissions of that role.

There are four default user roles: Administrators, Code Deployers, Operators, and Viewers. In addition, you can create custom roles.

For example, you might want to create a user role that grants users permission to view but not edit a specific subset of node groups. Or you might want to divide up administrative privileges so that one user role is able to reset passwords while another can edit roles and create users.

Permissions are additive. If a user is associated with multiple roles, that user is able to perform all of the actions described by all of the permissions on all of the applied roles.

Structure of user permissions

User permissions are structured as a triple of *type*, *permission*, and *object*.

- **Types** are everything that can be acted on, such as node groups, users, or user roles.
- **Permissions** are what you can do with each type, such as create, edit, or view.

- **Objects** are specific instances of the type.

Some permissions added to the Administrators user role might look like this:

Type	Permission	Object	Description
Node groups	View	PE Master	Gives permission to view the PE Master node group.
User roles	Edit	All	Gives permission to edit all user roles.

When no object is specified, a permission applies to all objects of that type. In those cases, the object is “All”. This is denoted by “*” in the API.

In both the console and the API, “*” is used to express a permission for which an object doesn’t make sense, such as when creating users.

User permissions

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

Note that types and permissions have both a display name, which is the name you see in the console interface, and a system name, which is the name you use to construct permissions in the API. In the following table, the display names are used.

Type	Permission	Definition
Certificate request	Accept and reject	Accept and reject certificate signing requests. Object must always be “*”.
Console	View	View the PE console. Object must always be “*”.
Directory service	View, edit, and test	View, edit, and test directory service settings. Object must always be “*”.
Job orchestrator	Start, stop and view jobs	Start and stop jobs and tasks, view jobs and job progress, and view an inventory of nodes that are connected to the PCP broker.
Node groups	Create, edit, and delete child groups	Create new child groups, delete existing child groups, and modify every attribute of child groups except environment. This permission is inherited by all descendents of the node group.
Node groups	Edit child group rules	Edit the rules of descendents of a node group. This does not grant the ability to edit the rules of the group in the object field, only children of that group. This permission is inherited by all descendents of the node group.
Node groups	Edit classes, parameters, and variables	Edit every attribute of a node group except its environment and rule. This permission is inherited by all descendents of the node group.

Type	Permission	Definition
Node groups	Edit configuration data	Edit parameterized configuration data on a node group. This permission is inherited by all descendents of the node group.
Node groups	Edit parameters and variables	Edit the class parameters and variables of a node group's classes. This permission is inherited by all descendents of the node group.
Node groups	Set environment	Set the environment of a node group. This permission is inherited by all descendents of the node group.
Node groups	View	See all attributes of a node group, most notably the values of class parameters and variables. This permission is inherited by all descendents of the node group.
Nodes	Edit node data from PuppetDB	Edit node data imported from PuppetDB. Object must always be " * ".
Nodes	View node data from PuppetDB	View node data imported from PuppetDB. Object must always be " * ".
Nodes	View sensitive connection information in inventory service	View sensitive parameters stored in the inventory service for a connection. For example, user credentials. Object must always be " * ".
Plans	Run plans	Run specific plans on all nodes.
Puppet agent	Run Puppet on agent nodes	Trigger a Puppet run from the console or orchestrator. Object must always be " * ".
Puppet environment	Deploy code	Deploy code to a specific PE environment.
Puppet Server	Compile catalogs for remote nodes	Compile a catalog for any node managed by this PE instance. This permission is required to run impact analysis tasks in Continuous Delivery for Puppet Enterprise.

Type	Permission	Definition
Tasks	Run tasks	Run specific tasks on all nodes, a selected node group, or nodes that match a PQL query. Important: A task must be permitted to run on all nodes in order to run on nodes that are outside of the PuppetDB (over SSH or WinRM for example). As a result, users with such permissions can run tasks on any nodes they have the credentials to access.
User groups	Delete	Delete a user group. This can be granted per group.
User groups	Import	Import groups from the directory service for use in RBAC. Object must always be " * ".
User roles	Create	Create new roles. Object must always be " * ".
User roles	Edit	Edit and delete a role. Object must always be " * ".
User roles	Edit members	Change which users and groups a role is assigned to. This can be granted per role.
Users	Create	Create new local users. Remote users are "created" by that user authenticating for the first time with RBAC. Object must always be " * ".
Users	Edit	Edit a local user's data, such as name or email, and delete a local or remote user from PE. This can be granted per user.
Users	Reset password	Grant password reset tokens to users who have forgotten their passwords. This process also reinstates a user after the use has been revoked. This can be granted per user.
Users	Revoke	Revoke or disable a user. This means the user is no longer able to authenticate and use the console, node classifier, or RBAC. This permission also includes the ability to revoke the user's authentication tokens. This can be granted per user.

Display names and corresponding system names

The following table provides both the display and system names for the types and all their corresponding permissions.

Type (display name)	Type (system name)	Permission (display name)	Permission (system name)
Certificate requests	cert_requests	Accept and reject	accept_reject
Console	console_page	View	view
Directory service	directory_service	View, edit, and test	edit
Job orchestrator	orchestrator	Start, stop and view jobs	view
Node groups	node_groups	Create, edit, and delete child groups	modify_children
Node groups	node_groups	Edit child group rules	edit_child_rules
Node groups	node_groups	Edit classes, parameters, and variables	edit_classification
Node groups	node_groups	Edit configuration data	edit_config_data
Node groups	node_groups	Edit parameters and variables	edit_params_and_vars
Node groups	node_groups	Set environment	set_environment
Node groups	node_groups	View	view
Nodes	nodes	Edit node data from PuppetDB	edit_data
Nodes	nodes	View node data from PuppetDB	view_data
Nodes	nodes	View sensitive connection information in inventory service	view_inventory_sensitive
Plans	plans	Run Plans	run
Puppet agent	puppet_agent	Run Puppet on agent nodes	run
Puppet environment	environment	Deploy code	deploy_code
Puppet Server	puppetserver	Compile catalogs for remote nodes	compile_catalogs
Tasks	tasks	Run Tasks	run
User groups	user_groups	Import	import
User roles	user_roles	Create	create
User roles	user_roles	Edit	edit
User roles	user_roles	Edit members	edit_members
Users	users	Create	create
Users	users	Edit	edit
Users	users	Reset password	reset_password
Users	users	Revoke	disable

Related information

[Permissions endpoints](#) on page 237

You assign permissions to user roles to manage user access to objects. The permissions endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

Working with node group permissions

Node groups in the node classifier are structured hierarchically; therefore, node group permissions inherit. Users with specific permissions on a node group implicitly receive the permissions on any child groups below that node group in the hierarchy.

Two types of permissions affect a node group: those that affect a group itself, and those that affect the group's child groups. For example, giving a user the "Set environment" permission on a group allows the user to set the environment for that group and all of its children. On the other hand, assigning "Edit child group rules" to a group allows a user to edit the rules for any child group of a specified node group, but not for the node group itself. This allows some users to edit aspects of a group, while other users can be given permissions for all children of that group without being able to affect the group itself.

Due to the hierarchical nature of node groups, if a user is given a permission on the default (All) node group, this is functionally equivalent to giving them that permission on " * ".

Best practices for assigning permissions

Working with user permissions can be a little tricky. You don't want to grant users permissions that essentially escalate their role, for example. The following sections describe some strategies and requirements for setting permissions.

Grant edit permissions to users with create permissions

Creating new objects doesn't automatically grant the creator permission to view those objects. Therefore, users who have permission to create roles, for example, must also be given permission to edit roles, or they won't be able to see new roles that they create. Our recommendation is to assign users permission to edit all objects of the type that they have permission to create. For example:

Type	Permission	Object
User roles	Edit members	All (or " * ")
Users	Edit	All (or " * ")

Avoid granting overly permissive permissions

Operators, a default role in PE, have many of the same permissions as Administrators. However, we've intentionally limited this role's ability to edit user roles. This way, members of this group can do many of the same things as Administrators, but they can't edit (or enhance) their own permissions.

Similarly, avoid granting users more permissions than their roles allow. For example, if users have the `roles:edit:*` permission, they are able to add the `node_groups:view:*` permission to the roles they belong to, and subsequently see all node groups.

Give permission to edit directory service settings to the appropriate users

The directory service password is not redacted when the settings are requested in the API. Give `directory_service:edit:*` permission only to users who are allowed see the password and other settings.

The ability to reset passwords should be given only with other password permissions

The ability to help reset passwords for users who forgot them is granted by the `users:reset_password:<instance>` permission. This permission has the side effect of reinstating revoked

users after the reset token is used. As such, the reset password permission should be given only to users who are also allowed to revoke and reinstate other users.

Creating and managing local users and user roles

Puppet Enterprise's role-based access control (RBAC) enables you to manage users—what they can create, edit, or view, and what they can't—in an organized, high-level way that is vastly more efficient than managing user permissions on a per-user basis. User roles are sets of permissions you can apply to multiple users. You can't assign permissions to single users in PE, only to user roles.

Remember: Each user must be assigned to one or more roles before they can log in and use PE.

Note: Puppet stores local accounts and directory service integration credentials securely. Local account passwords are hashed using SHA-256 multiple times along with a 32-bit salt. Directory service lookup credentials configured for directory lookup purposes are encrypted using AES-128. Puppet does not store the directory credentials used for authenticating to Puppet. These are different from the directory service lookup credentials.

Create a new user

These steps add a local user.

To add users from an external directory, see [Working with user groups from an external directory](#).

1. In the console, on the **Access control** page, click the **Users** tab.
2. In the **Full name** field, enter the user's full name.
3. In the **Login** field, enter a username for the user.
4. Click **Add local user**.

Give a new user access to the console

When you create new local users, you need to send them a password reset token so that they can log in for the first time.

1. On the **Access control** page, on the **Users** tab, select the user's full name.
2. Click **Generate password reset**.
3. Copy the link provided in the message and send it to the new user.

Create a new user role

RBAC has four predefined roles: Administrators, Code Deployers, Operators, and Viewers. You can also define your own custom user roles.

Users with the appropriate permissions, such as Administrators, can define custom roles. To avoid potential privilege escalation, only users who are allowed all permissions should be given the permission to edit user roles.

1. In the console, on the **Access control** page, click the **User roles** tab.
2. In the **Name** field, enter a name for the new user role.
3. (Optional) In the **Description** field, enter a description of the new user role.
4. Click **Add role**.

Assign permissions to a user role

You can mix and match permissions to create custom user roles that provide users with precise levels of access to PE actions.

Before you begin

Review [User permissions and user roles](#), which includes important information about how permissions work in PE.

1. On the **Access control** page, on the **User roles** tab, select a user role.

2. Click **Permissions**.
3. In the **Type** field, select the type of object you want to assign permissions for, such as **Node groups**.
4. In the **Permission** field, select the permission you want to assign, such as **View**.
5. In the **Object** field, select the specific object you want to assign the permission to. For example, if you are setting a permission to view node groups, select a specific node group this user role has permissions to view.
6. Click **Add permission**, and commit changes.

Related information

[Best practices for assigning permissions](#) on page 201

Working with user permissions can be a little tricky. You don't want to grant users permissions that essentially escalate their role, for example. The following sections describe some strategies and requirements for setting permissions.

Add a user to a user role

When you add users to a role, the user gains the permissions that are applied to that role. A user can't do anything in PE until they have been assigned to a role.

1. On the **Access control** page, on the **User roles** tab, select a user role.
2. Click **Member users**.
3. In the **User name** field, select the user you want to add to the user role.
4. Click **Add user**, and commit changes.

Remove a user from a user role

You can change a user's permissions by removing them from a user role. The user loses the permissions associated with the role, and won't be able to do anything in PE until they are assigned to a new role.

1. On the **Access control** page, on the **User roles** tab, select a user role.
2. Click **Member users**.
3. Locate the user you want to remove from the user role. Click **Remove**, and commit changes.

Revoke a user's access

If you want to remove a user's access to PE but not delete their account, you can revoke them. Revocation is also what happens when a user is locked out from too many incorrect password attempts.

1. In the console, on the **Access control** page, click the **Users** tab.
2. In the **Full name** column, select the user you want to revoke.
3. Click **Revoke user access**.

Tip: To unrevoke a user, follow the steps above and click **Reinstate user access**.

Change account expiration settings

You can specify the number of days before an inactive user's account is automatically revoked and update how often PE checks for idle user accounts.

`rbac_account_expiry_days`

The `rbac_account_expiry_days` parameter is a positive integer that specifies the duration, in days, before an inactive user account expires. If a non-superuser hasn't logged in to the console during the specified period, their user status updates to revoked. The default value is undefined.

Important: If the `account-expiry-days` parameter is not specified, or has a value of less than 1 (day), the `account-expiry-check-minutes` parameter is ignored.

To activate this setting in the console, add a value of 1 (day) or greater to the `rbac_account_expiry_days` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** group.

rbac_account_expiry_check_minutes

The `rbac_account_expiry_check_minutes` parameter is a positive integer that specifies how often, in minutes, the application checks for idle user accounts. The default value is 60 (minutes).

To change this setting in the console, set a value (in minutes) for the `rbac_account_expiry_check_minutes` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** group.

For more information on configuration options, or to use a different configuration method for changing settings, see [Methods for configuring PE](#) on page 160.

Delete a user

You can delete a user through the console. Note, however, that this action deletes only the user's Puppet Enterprise account, not the user's listing in any external directory service.

Deletion removes all data about the user except for their activity data, which continues to be stored in the database and remains viewable through the API.

1. In the console, on the **Access control** page, click the **Users** tab.
2. In the **Full name** column, locate the user you want to delete.
3. Click **Remove**.

Note: Users with superuser privileges cannot be deleted, and the **Remove** button does not appear for these users.



CAUTION: If a user is deleted from the console and then recreated with the same full name and login, PE issues the recreated user a new unique user ID. In this instance, queries for the login to the API's activity database return information on both the deleted user and the new user. However, in the console, the new user's **Activity** tab does not display information about the deleted user's account.

Delete a user role

You can delete a user role through the console.

When you delete a user role, users lose the permissions that the role gives them. This can impact their access to Puppet Enterprise if they have not been assigned other user roles.

1. In the console, on the **Access control** page, click the **User roles** tab.
2. In the **Name** column, locate the role you want to delete.
3. Click **Remove**.

Connecting LDAP external directory services to PE

Puppet Enterprise connects to external Lightweight Directory Access Protocol (LDAP) directory services through its role-based access control (RBAC) service. Because PE integrates with cloud LDAP service providers such as Okta, you can use existing users and user groups that have been set up in your external directory service.

Specifically, you can:

- Authenticate external directory users.
- Authorize access of external directory users based on RBAC permissions.
- Store and retrieve the groups and group membership information that has been set up in your external directory.

Note: Puppet stores local accounts and directory service integration credentials securely. Local account passwords are hashed using SHA-256 multiple times along with a 32-bit salt. Directory service lookup credentials configured for directory lookup purposes are encrypted using AES-128. Puppet does not store the directory credentials used for authenticating to Puppet. These are different from the directory service lookup credentials.

PE supports OpenLDAP and Active Directory. If you have predefined groups in OpenLDAP or Active Directory, you can import these groups into the console and assign user roles to them. Users in an imported group inherit the

permissions specified in assigned user roles. If new users are added to the group in the external directory, they also inherit the permissions of the role to which that group belongs.

Note: The connection to OpenLDAP and Active Directory is read-only. If you want to make changes to remote users or user groups, you need to edit the information directly in the external directory.

Connect to an external directory service

PE connects to the external directory service when a user logs in or when groups are imported. The supported directory services are OpenLDAP and Active Directory.

1. In the console, on the **Access control** page, click the **LDAP** tab.
2. Fill in the directory information.

All fields are required, except for **Login help**, **Lookup user**, **Lookup password**, **User relative distinguished name**, and **Group relative distinguished name**.

If you do not enter **User relative distinguished name** or **Group relative distinguished name**, RBAC searches the entire base DN for the user or group.

3. Click **Test connection** to ensure that the connection has been established. Save your settings after you have successfully tested them.

Note: This only tests the connection to the LDAP server. It does not test or validate LDAP queries.

External directory settings

The table below provides examples of the settings used to connect to an Active Directory service and an OpenLDAP service to PE. Each setting is explained in more detail below the table.

Important: The settings shown in the table are examples. You need to substitute these example settings with the settings used in your directory service.

Name	Example Active Directory settings	Example OpenLDAP settings
Directory name	My Active Directory	My Open LDAP Directory
Login help (optional)	https://myweb.com/ldaploginhelp	https://myweb.com/ldaploginhelp
Hostname	myhost.delivery.exampleservice.net	myhost.delivery.exampleservice.net
Port	389 (636 for LDAPS)	389 (636 for LDAPS)
Lookup user (optional)	cn=queryuser,cn=Users,dc=puppetlabs,dc=net	cn=admin,dc=delivery,dc=puppetlabs,dc=net
Lookup password (optional)	The lookup user's password.	The lookup user's password.
Connection timeout (seconds)	10	10
Connect using:	SSL	StartTLS
Validate the hostname?	Default is yes.	Default is yes.
Allow wildcards in SSL certificate?	Default is no.	Default is no.
Base distinguished name	dc=puppetlabs,dc=com	dc=puppetlabs,dc=com
User login attribute	sAMAccountName	cn
User email address	mail	mail
User full name	displayName	displayName
User relative distinguished name (optional)	cn=users	ou=users
Group object class	group	groupOfUniqueNames
Group membership field	member	uniqueMember

Name	Example Active Directory settings	Example OpenLDAP settings
Group name attribute	name	displayName
Group lookup attribute	cn	cn
Group relative distinguished name (optional)	cn=groups	ou=groups
Turn off LDAP_MATCHING_RULE_IN_CHAIN?	Default is no.	Default is no.
Search nested groups?	Default is no.	Default is no.

Explanation of external directory settings

Directory name The name that you provide here is used to refer to the external directory service anywhere it is used in the PE console. For example, when you view a remote user in the console, the name that you provide in this field is listed in the console as the source for that user. Set any name of your choice.

Login help (optional) If you supply a URL here, a "Need help logging in?" link is displayed on the login screen. The href attribute of this link is set to the URL that you provide.

Hostname The FQDN of the directory service to which you are connecting.

Port The port that PE uses to access the directory service. The port is generally 389, unless you choose to connect using SSL, in which case it is generally 636.

Lookup user (optional) The distinguished name (DN) of the directory service user account that PE uses to query information about users and groups in the directory server. If a username is supplied, this user must have read access for all directory entries that are to be used in the console. We recommend that this user is restricted to read-only access to the directory service.

If your LDAP server is configured to allow anonymous binding, you do not need to provide a lookup user. In this case, the RBAC service binds anonymously to your LDAP server.

Lookup password (optional) The lookup user's password.

If your LDAP server is configured to allow anonymous binding, you do not need to provide a lookup password. In this case, the RBAC service binds anonymously to your LDAP server.

Connection timeout (seconds) The number of seconds that PE attempts to connect to the directory server before timing out. Ten seconds is fine in the majority of cases. If you are experiencing timeout errors, make sure the directory service is up and reachable, and then increase the timeout if necessary.

Connect using: Select the security protocol you want to use to connect to the external directory: **SSL** and **StartTLS** encrypt the data transmitted. **Plain text** is not a secure connection. In addition, to ensure that the directory service is properly identified, configure the `ds-trust-chain` to point to a copy of the public key for the directory service. For more information, see [Verify directory server certificates](#) on page 208.

Validate the hostname? Select **Yes** to verify that the Directory Services hostname used to connect to the LDAP server matches the hostname on the SSL certificate. This option is not available when you choose to connect to the external directory using plain text.

Allow wildcards in SLL certificate? Select **Yes** to allow a connection to a Directory Services server with a SSL certificates that use a wildcard (*) specification. This option is not available when you choose to connect to the external directory using plain text.

Base distinguished name When PE constructs queries to your external directory (for example to look up user groups or users), the queries consist of the relative distinguished name (RDN) (optional) + the base distinguished name (DN), and are then filtered by lookup/login attributes. For example, if PE wants to authenticate a user named Bob who has the RDN `ou=bob, ou=users`, it sends a query in which the RDN is concatenated with the DN specified in this field (for example, `dc=puppetlabs, dc=com`). This gives a search base of `ou=bob, ou=users, dc=puppetlabs, dc=com`.

The base DN that you provide in this field specifies where in the directory service tree to search for groups and users. It is the part of the DN that all users and groups that you want to use have in common. It is commonly the root DN (example `dc=example,dc=com`) but in the following example of a directory service entry, you could set the base DN to `ou=Puppet,dc=example,dc=com` because both the group and the user are also under the organizational unit `ou=Puppet`.

Example directory service entry

```
# A user named Harold
dn: cn=harold,ou=Users,ou=Puppet,dc=example,dc=com
objectClass: organizationalPerson
cn: harold
displayName: Harold J.
mail: harold@example.com
memberOf: inspectors
sAMAccountName: harold11

# A group Harold is in
dn: cn=inspectors,ou=Groups,ou=Puppet,dc=example,dc=com
objectClass: group
cn: inspectors
displayName: The Inspectors
member: harold
```

User login attribute This is the directory attribute that the user uses to log in to PE. For example, if you specify `sAMAccountName` as the user login attribute, Harold logs in with the username "harold11" because `sAMAccountName=harold11` in the example directory service entry provided above.

The value provided by the user login attribute must be unique among all entries under the User RDN + Base DN search base you've set up.

For example, say you've selected the following settings:

```
base DN = dc=example,dc=com
user RDN = null
user login attribute = cn
```

When Harold tries to log in, the console searches the external directory for any entries under `dc=example,dc=com` that have the attribute/value pair `cn=harold`. (This attribute/value pair does not need to be contained within the DN). However, if there is another user named Harold who has the DN `cn=harold,ou=OtherUsers,dc=example,dc=com`, two results are returned and the login does not succeed because the console does not know which entry to use. Resolve this issue by either narrowing your search base such that only one of the entries can be found, or using a value for login attribute that you know to be unique. This makes `sAMAccountName` a good choice if you're using Active Directory, as it must be unique across the entire directory.

User email address The directory attribute to use when displaying the user's email address in PE.

User full name The directory attribute to use when displaying the user's full name in PE.

User relative distinguished name (optional) The user RDN that you set here is concatenated with the base DN to form the search base when looking up a user. For example, if you specify `ou=users` for the user RDN, and your base DN setting is `ou=Puppet,dc=example,dc=com`, PE finds users that have `ou=users,ou=Puppet,dc=example,dc=com` in their DN.

This setting is optional. If you choose not to set it, PE searches for the user in the base DN (example: `ou=Puppet,dc=example,dc=com`). Setting a user RDN is helpful in the following situations:

- When you experience long wait times for operations that contact the directory service (either when logging in or importing a group for the first time). Specifying a user RDN reduces the number of entries that are searched.
- When you have more than one entry under your base DN with the same login value.

Tip: It is not currently possible to specify multiple user RDNs. If you want to filter RDNs when constructing your query, we suggest creating a new lookup user who only has read access for the users and groups you want to use in PE.

Group object class The name of an object class that all groups have.

Group membership field Tells PE how to find which users belong to which groups. This is the name of the attribute in the external directory groups that indicates who the group members are.

Group name attribute The attribute that stores the display name for groups. This is used for display purposes only.

Group lookup attribute The value used to import groups into PE. Given the example directory service entry provided above, the group lookup attribute would be `cn`. When specifying the Inspectors group in the console to import it, provide the name `inspectors`.

The value for this attribute must be unique under your search base. If you have users with the same login as the lookup of a group that you want to use, you can narrow the search base, use a value for the lookup attribute that you know to be unique, or specify the **Group object class** that all of your groups have in common but your users do not.

Tip: If you have a large number of nested groups in your group hierarchy, or you experience slowness when logging in with RBAC, we recommend disabling nested group search unless you need it for your authorization schema to work.

Group relative distinguished name (optional) The group RDN that you set here is concatenated with the base DN to form the search base when looking up a group. For example, if you specify `ou=groups` for the group RDN, and your base DN setting is `ou=Puppet,dc=example,dc=com`, PE finds groups that have `ou=groups,ou=Puppet,dc=example,dc=com` in their DN.

This setting is optional. If you choose not to set it, PE searches for the group in the base DN (example: `ou=Puppet,dc=example,dc=com`). Setting a group RDN is helpful in the following situations:

- When you experience long wait times for operations that contact the directory service (either when logging in or importing a group for the first time). Specifying a group RDN reduces the number of entries that are searched.
- When you have more than one entry under your base DN with the same lookup value.

Tip: It is not currently possible to specify multiple group RDNs. If you want to filter RDNs when constructing your query, create a new lookup user who only has read access for the users and groups you plan to use in PE.

Note: At present, PE supports only a single Base DN. Use of multiple user RDNs or group RDNs is not supported.

Turn off LDAP_MATCHING_RULE_IN_CHAIN? Select **Yes** to turn off the LDAP matching rule that looks up the chain of ancestry for an object until it finds a match. For organizations with a large number of group memberships, matching rule in chain can slow performance.

Search nested groups? Select **Yes** to search for groups that are members of an external directory group. For organizations with a large number of nested group memberships, searching nested groups can slow performance.

Related information

[PUT /ds](#) on page 243

Replaces current directory service connection settings. Authentication is required.

Verify directory server certificates

To ensure that RBAC isn't being subjected to a Man-in-the Middle (MITM) attack, verify the directory server's certificate.

When you select SSL or StartTLS as the security protocol to use for communications between PE and your directory server, the connection to the directory is encrypted. To ensure that the directory service is properly identified, configure the `ds-trust-chain` to point to a copy of the public key for the directory service.

The RBAC service verifies directory server certificates using a trust store file, in Java Key Store (JKS), PEM, or PKCS12 format, that contains the chain of trust for the directory server's certificate. This file needs to exist on disk in a location that is readable by the user running the RBAC service.

To turn on verification:

1. In the console, click **Node groups**.
2. Open the **PE Infrastructure** node group and select the **PE Console** node group.
3. Click **Classes**. Locate the `puppet_enterprise::profile::console` class.
4. In the **Parameter** field, select `rbac_ds_trust_chain`.
5. In the **Value** field, set the absolute path to the trust store file.
6. Click **Add parameter**, and commit changes.
7. To make the change take effect, run Puppet. Running Puppet restarts pe-console-services.

After this value is set, the directory server's certificate is verified whenever RBAC is configured to connect to the directory server using SSL or StartTLS.

Enable custom password policies through LDAP

Password policies are not configurable in PE. However, if your organization requires more complex password policies than the default, you can allow LDAP to manage custom password policies by delegating administrative privileges to LDAP and then revoking the admin user in the console.

Before you begin

Enable LDAP by [Connecting LDAP external directory services to PE](#) on page 204. Make sure you are logged into LDAP as the administrative user.

Revoke the admin user:

1. In the console, on the **Access control** page, click the **Users** tab.
2. Select **Administrator**.
3. On the **User details** page, click **Revoke user access**.

You have revoked the admin user in the console, which allows LDAP to manage password policies. To enable the admin again, select **Reinstate user access** on the admin's **User details** page.

Working with user groups from a LDAP external directory

You don't explicitly add remote users to PE. Instead, after the external directory service has been successfully connected, remote users must log into PE, which creates their user record.

If the user belongs to an external directory group that has been imported into PE and then assigned to a role, the user is assigned to that role and gains the privileges of the role. Roles are additive: You can assign users to more than one role, and they gain the privileges of all the roles to which they are assigned.

Import a user group from an external directory service

You import existing external directory groups to PE explicitly, which means you add the group by name.

1. In the console, on the **Access control** page, click the **User groups** tab.

User groups is available only if you have established a connection with an external directory.

2. In the **Login** field, enter the name of a group from your external directory.
3. Click **Add group**.

Remember: No user roles are listed until you add this group to a role. No users are listed until a user who belongs to this group logs into PE.

Troubleshooting: A PE user and user group have the same name

If you have both a PE user and an external directory user group with the exact same name, PE throws an error when you try to log on as that user or import the user group.

To work around this problem, you can change your settings to use different RDNs for users and groups. This works as long as all of your users are contained under one RDN that is unique from the RDN that contains all of your groups.

Assign a user group to a user role

After you've imported a group, you can assign it a user role, which gives each group member the permissions associated with that role. You can add user groups to existing roles, or you can create a new role, and then add the group to the new role.

1. In the console, on the **Access control** page, click the **User roles** tab.
2. Click the role you want to add the user group to.
3. Click **Member groups**. In the **Group name** field, select the user group you want to add to the user role.
4. Click **Add group**, and commit changes.

Delete a user group

You can delete a user group in the console. Users who were part of the deleted group lose the permissions associated with roles assigned to the group.

Remember: This action removes the group only from Puppet Enterprise, not from the associated external directory service.

1. In the console, on the **Access control** page, click the **User groups** tab.
User groups is available only if you have established a connection with an external directory.
2. Locate the group that you wish to delete.
3. Click **Remove**.

Removing a remote user's access to PE

In order to fully revoke the remote user's access to Puppet Enterprise, you must also remove the user from the external directory groups accessed by PE.

Deleting a remote user's PE account does not automatically prevent that user from accessing PE in the future. So long as the remote user is still a member of a group in an external directory that PE is configured to access, the user retains the ability to log into PE.

If you delete a user from your external directory service but not from PE, the user can no longer log in, but any generated tokens or existing console sessions continue to be valid until they expire. To invalidate the user's tokens or sessions, revoke the user's PE account, which also automatically revokes all tokens for the user. You must manually delete the user for their account record to disappear.

Connect a SAML identity provider to PE

Connect to a Security Assertion Markup Language (SAML) identity provider to log in to PE with single sign-on (SSO). SSO authentication securely centralizes sensitive data and reduces the number of login credentials users have to remember and store. Depending on your identity provider, you can also use this workflow to connect and configure multifactor authentication (MFA) in PE.

Note: When SAML is configured, you must generate a token in the console to use CLI tools like orchestrator jobs or PuppetDB queries triggered from the command line.

An *identity provider (IdP)* is a service that stores and maintains user information under a single login. Okta, PingID, and Salesforce are all SAML identity providers.

The identity provider sends an *assertion*, or an xml document containing the required attributes for authenticating the user, to the service provider. *Attributes* are name/value pairs that specify pieces of information about a user, like their email or name.

The *service provider* receives the assertion from the identity provider via the web and confirms the attributes match the user, who then is logged into the website or application. PE is a service provider.

After connecting a SAML identity provider to PE, you can log into and out of PE through the identity provider.

Note: When using encryption with SAML, users might experience delays and timeouts when logging out if the host system that runs PE lacks sufficient entropy.

Get URLs and the signing and encryption certificate

PE provides URLs and a certificate that you must configure in your identity provider before you can configure SSO in PE. You must use the console to view the URLs and certificate if you haven't configured SSO yet. After you've configured SSO, you can retrieve them using the [GET /v1/saml/meta](#) on page 245 endpoint. If you're promoting a replica, you must specify your replica's new URLs and certificate in your IdP's configuration.

1. In the console, on the **Access control** page, click the **SSO** tab.
2. Click **Show configuration information**.
The following configuration values are populated:
 - The SAML metadata URL
 - The assertion consumer service URL
 - The single logout service URL
 - The signing and encryption certificate
3. Add the URLs and certificate where appropriate to your identity provider configuration.

Attribute binding

Attribute binding links attribute names from PE to attributes in the identity provider. When configuring SSO, choose the name of the attributes for PE and map them to the corresponding values in your identity provider configuration.

There are no standard SAML attribute names, but attribute binding ensures PE and your identity provider can identify attributes from one another without having to call them the same thing. This capability allows you to connect PE to a variety of different identity providers.

For example, you might want to name the User attribute “uid” in PE, which corresponds to a unique user ID. When you configure attribute binding with your identity provider, map “uid” to the corresponding value your identity provider uses to identify the unique user ID, for example, “user.login”.

After configuring attribute binding for User in PE and in your identity provider, any time PE receives an assertion from the identity provider, it knows that “user.login” is the same thing as “uid”, and vice versa.

If you are connected to a LDAP external directory service, consider using the same attribute names you use in your LDAP configuration.

Attribute binding occurs for four attributes:

User

The login field that consistently identifies a given user across multiple platforms. If migrating from LDAP, this is the same as the "user login field".

Example: "uid"

Email

Extracts the email address of the user.

Example: "email"

Display name

Displays a friendly name for the user, usually the first and last name.

Example: "name"

Groups

Automatically associates the user groups and their assigned roles in PE. The attribute maps to the "login" value of the user group.

Example: "group"

Note: Some identity providers might not use the term "attribute" or the phrase "attribute binding" when referring to the name/value pairs.

Connect to a SAML identity provider

Use the console to set up SSO or MFA with your SAML identity provider.

Before you begin

Add URLs and the encryption and signing certificate to your identity provider configuration.

Configure attribute binding in your identity provider.

Configure users and user groups with your identity provider.

1. In the console, on the **Access control** page, click the **SSO** tab.

2. Click **Configure**.

3. Fill in the configuration information.

All fields are required except **Identity provider SLO response URL**, and the fields in the **Contacts** and **Organization** sections.

4. Commit changes.

Related information

[Promote a replica](#) on page 192

If your primary server can't be restored, you can promote the replica to primary server to establish the replica as the new, permanent primary server.

Generate a token in the console

Use the console to generate an authentication token and use it to access APIs. If SAML is configured, you must have a token to use CLI tools like orchestrator jobs or PuppetDB queries triggered from the command line. Generate and export a token to the machine you want to run the CLI tool on.

1. In the console, on the **My account** page, click the **Tokens** tab.

2. Click **Generate new token**.

3. Under **Description**, enter a description for your new token.

4. Under **Lifetime**, select the length of time you want your token to be good for.

5. Click **Get token**.

6. Click **Copy token**.

Important: Store the token somewhere secure and do not share it with others. You cannot regenerate this token again once you close this page.

7. Click **Close**.

SAML configuration reference

Configure these settings in PE and your SAML identity provider to enable SSO or MFA.

Setting	Maps to	Required or optional	Definition
Display name	display_name	Required	A required string that identifies the IdP used for SSO or MFA login. Example: "Corporate Okta"

Setting	Maps to	Required or optional	Definition
Identity provider entity ID	idp_entity_id	Required	A URL string identifying your IdP. Your IdP's configuration has this information. Example: "https://sso.example.info/entity"
Identity provider SSO URL	idp_sso_url	Required	The URL PE sends authentication messages to. Your IdP configuration has this information. Example: "https://idp.example.org/SAML2/SSO"
Identity provider SLO URL	idp_slo_url	Required	The URL PE sends the single logout request (SLO) to. Your IdP configuration has this information. Example: "https://ipd.example.com/SAML2/SLO"
Identity provider SLO response URL	idp_slo_response_url	Optional	An optional URL specifying an alternative location for SLO handling. Defaults to the SLO URL. Example: "https://ipd.example.com/SAML2/SLO-response"

Setting	Maps to	Required or optional	Definition
IdP certificate	idp_certificate	Required	<p>The public x509 certificate of the identity provider, in PEM format. PE uses the certificate to decrypt messages from the IdP and validate signatures.</p> <p>Example:</p> <pre>-----BEGIN CERTIFICATE----- MIIGADCCA +igAwIBAgIBAjANBgkqhkiG9w0BAQsF ... STkGww== -----END CERTIFICATE-----</pre>
Name ID encrypted?	name_id_encrypted	Optional	<p>Indicates whether you want PE to encrypt the name-id in the logout request.</p> <p>Boolean, defaults to <code>true</code>.</p>
Sign authentication requests?	authn_request_signed	Optional	<p>Indicates whether you want PE to cryptographically sign authentication request it sends to the IdP.</p> <p>Boolean, defaults to <code>true</code>.</p>
Sign logout response?	logout_response_signed	Optional	<p>Indicates whether you want PE to cryptographically sign logout responses it sends to the IdP.</p> <p>Boolean, defaults to <code>true</code>.</p>
Sign logout requests?	logout_request_signed	Optional	<p>Indicates whether you want PE to cryptographically sign logout requests it sends to the IdP.</p> <p>Boolean, defaults to <code>true</code>.</p>

Setting	Maps to	Required or optional	Definition
Require signed messages?	want_messages_signed	Required	Indicates whether you want the IdP to cryptographically sign all responses, logout requests, and logout response messages it sends to PE. Boolean, defaults to true.
Require signed assertions?	want_assertions_signed	Required	Indicates whether you want the IdP to cryptographically sign assertion elements it sends to PE. Boolean, defaults to true.
Sign metadata?	sign_metadata	Required	Indicates whether you want PE to cryptographically sign the metadata provided for the IdP configuration. Boolean, defaults to true.
Require encrypted assertions?	want_assertions_encrypted	Required	Indicates whether you want the IdP to encrypt the assertion messages it sends to PE. Boolean, defaults to true.
Require name ID encrypted?	want_name_id_encrypted	Required	Indicates whether you want the IdP to encrypt the name-id field in messages it sends to PE. Boolean, defaults to true.
Requested authentication context	requested_auth_context	Optional	Comma separated list of authentication contexts indicating the type of user authentication PE suggest to the IdP. Authentication types are defined in the urn:oasis:names:tc:SAML:2.0:ac:classes: namespace of the SAML specification. If left blank, no authentication context is sent. Example: urn:oasis:names:tc:SAML:2.0:ac:classes:password

Setting	Maps to	Required or optional	Definition
Requested authentication context comparison	requested_auth_context_comparison	Optional	<p>Indicates to the IdP the strength of the authentication context PE provides. Choose from one of the following:</p> <ul style="list-style-type: none"> • exact: The requested authentication context comparison must be an exact match with one of the contexts PE provides. • minimum (default): The requested authentication context comparison must be, at minimum, the strength of the context PE provides. • maximum: The requested authentication context comparison is, at most, equal to the context PE provides. • better: The requested authentication context comparison must be higher than the context PE provides.

Setting	Maps to	Required or optional	Definition
Validate xml?	want_xml_validation	Required	Indicates whether you want PE to validate all xml statements it receives from your IdP. Invalid xml might cause security issues. Boolean, defaults to true.
Signature algorithm	signature_algorithm	Required	Indicates which signing algorithm PE uses to sign messages. Choose from one of the following: <ul style="list-style-type: none"> rsa-sha1 dsa-sha1 rsa-sha256 (default) rsa-sha384 rsa-sha512
Organization name	organizational_name	Optional	The official name of your organization.
Organization display name	organizational_display_name	Optional	An alternative display name for your organization
Organization URL	organizational_url	Optional	The URL for your company.
Organizational language	organizational_language	Optional	The standard abbreviation for the preferred spoken language at your organization. Example: "en" for English
Technical contact name	technical_support_name	Optional	The name of the main technical contact at your organization.
Technical contact email address	technical_support_email	Optional	The email address for the main technical contact at your organization.
Support contact name	support_name	Optional	The name of the main support contact at your organization.
Support contact email address	support_email	Optional	The email address for the main support contact at your organization.

Token-based authentication

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

You can generate authentication tokens using the console, `puppet-access`, or the RBAC API V2 endpoint. You can also generate one-off tokens that do not need to be saved, which are typically used by a service.

You can use the console to view or revoke your tokens on the **My account** page, in the **Tokens** tab. Administrators can view and revoke tokens for other users on the **User details** page.

RBAC settings for authentication tokens can be managed in the console in the **PE Infrastructure** group. See [Configure RBAC and token-based authentication settings](#) on page 168 for a list of settings you can change.

Authentication tokens manage access to the following PE services:

- Puppet orchestrator
- Code Manager
- Node classifier
- Role-based access control (RBAC)
- Activity service
- PuppetDB

Related information

[Installing PE client tools](#) on page 138

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

[User permissions](#) on page 197

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

Configure puppet-access

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. If you want to use `puppet-access`, ensure it is configured correctly before using it to generate authentication tokens.

The configuration file for `puppet-access` allows you to generate tokens from the CLI without having to pass additional flags.

Whether you are running `puppet-access` on a PE-managed server or installing it on a separate work station, you need a global configuration file and a user-specified configuration file.

Global configuration file

The global configuration file is located at:

- **On *nix systems:** `/etc/puppetlabs/client-tools/puppet-access.conf`
- **On Windows systems:** `C:/ProgramData/PuppetLabs/client-tools/puppet-access.conf`

On machines managed by Puppet Enterprise, this global configuration file is created for you. The configuration file is formatted in JSON. For example:

```
{
  "service-url": "https://<CONSOLE_HOSTNAME>:4433/rbac-api",
  "token-file": "~/.puppetlabs/token",
  "certificate-file": "/etc/puppetlabs/puppet/ssl/certs/ca.pem"
}
```

PE determines and adds the `service-url` setting.

If you're running `puppet-access` from a workstation not managed by PE, you must create the global file and populate it with the configuration file settings.

User-specified configuration file

The user-specified configuration file is located at `~/.puppetlabs/client-tools/puppet-access.conf` for both *nix and Windows systems. You must create the user-specified file and populate it with the configuration file settings. A list of configuration file settings is found in the next section.

The user-specified configuration file always takes precedence over the global configuration file. For example, if the two files have contradictory settings for the `token-file`, the user-specified settings prevail.

Important: User-specified configuration files must be in JSON format; HOCON and INI-style formatting are not supported.

Configuration file settings for puppet-access

As needed, you can manually add configuration settings to your user-specified or global `puppet-access` configuration file.

The class that manages the global configuration file is `puppet_enterprise::profile::controller`.

You can also change configuration settings by specifying flags when using the `puppet-access` command in the command line.

Setting	Description	Command line flag
<code>token-file</code>	The location for storing authentication tokens. Defaults to <code>~/.puppetlabs/token</code> .	<code>-t, --token-file</code>
<code>certificate-file</code>	The location of the CA that signed the console-services server's certificate. Defaults to the PE CA cert location, <code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code> .	<code>--ca-cert</code>
<code>config-file</code>	Changes the location of your configuration file. Defaults to <code>~/.puppetlabs/client-tools/puppet-access.conf</code> .	<code>-c, --config-file</code>
<code>service-url</code>	The URL for your RBAC API. Defaults to the URL automatically determined during the client tools package installation process, generally <code>https://<CONSOLE_HOSTNAME>:4433/rbac-api</code> . You typically need to change this only if you are moving the console server.	<code>--service-url</code>

Generate a token in the console

Use the console to generate an authentication token and use it to access APIs. If SAML is configured, you must have a token to use CLI tools like `orchestrator jobs` or `PuppetDB` queries triggered from the command line. Generate and export a token to the machine you want to run the CLI tool on.

1. In the console, on the **My account** page, click the **Tokens** tab.
2. Click **Generate new token**.

3. Under **Description**, enter a description for your new token.
4. Under **Lifetime**, select the length of time you want your token to be good for.
5. Click **Get token**.
6. Click **Copy token**.

Important: Store the token somewhere secure and do not share it with others. You cannot regenerate this token again once you close this page.

7. Click **Close**.

Generate a token using puppet-access

Use `puppet-access` to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Before you begin

Install the PE client tools package and configure `puppet-access`.

1. Choose one of the following options, depending on how long you need your token to last:
 - To generate a token with the default one-hour lifetime, run `puppet-access login`.
 - To generate a token with a specific lifetime, run `puppet-access login --lifetime <TIME PERIOD>`.

For example, `puppet-access login --lifetime 5h` generates a token that lasts five hours.

2. When prompted, enter the same username and password that you use to log into the PE console.
The `puppet-access` command contacts the token endpoint in the RBAC v1 API. If your login credentials are correct, the RBAC service generates a token.
3. The token is generated and stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. You can print the token at any time using `puppet-access show`.

You can continue to use this token until it expires, or until your access is revoked. The token has the exact same set of permissions as the user that generated it.



CAUTION: If you run the login command with the `--debug` flag, the client outputs the token, as well as the username and password. For security reasons, exercise caution when using the `--debug` flag with the login command.

Note: If a remote user generates a token and then is deleted from your external directory service, the deleted user cannot log into the console. However, because the token has already been authenticated, the RBAC service does not contact the external directory service again when the token is used in the future. To fully remove the token's access, you need to manually revoke or delete the user from PE.

Related information

[Set a token-specific lifetime](#) on page 222

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

Generate a token using the API endpoint

The RBAC v1 API includes a token endpoint, which allows you to generate a token using curl commands.

1. On the command line, post your RBAC user login credentials using the token endpoint.

```
type_header='Content-Type: application/json'
uri="https://$(puppet config print server):4433/rbac-api/v1/auth/token"
data='{ "login": "<USER>",
      "password": "<PASSWORD>" }'

curl --insecure --header "$type_header" --request POST "$uri" --data
"$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The command returns a JSON object containing the key `token` and the token itself.



CAUTION: If you are using curl commands with the `--k` or `--insecure` SSL connection options, keep in mind that you are vulnerable to a person-in-the-middle attack.

2. Save the token. Depending on your workflow, either:

- Copy the token to a text file.
- Save the token as an environment variable using `export TOKEN=<PASTE THE TOKEN HERE>`.

You can continue to use this token until it expires, or until your access is revoked. The token has the exact same set of permissions as the user that generated it.

Note: If a remote user generates a token and then is deleted from your external directory service, the deleted user cannot log into the console. However, because the token has already been authenticated, the RBAC service does not contact the external directory service again when the token is used in the future. To fully remove the token's access, you need to manually revoke or delete the user from PE.

Use a token with the PE API endpoints

The example below shows how to use a token in an API request. In this example, you use the `/users/current` endpoint of the RBAC v1 API to get information about the current authenticated user.

Before you begin

Generate a token using `puppet-access login`.

Run the following commands: `curl --insecure --header "X-Authentication: $(puppet-access show)" https://$(puppet config print server):4433/rbac-api/v1/users/current`

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current"

curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The commands above use the X-Authentication header to supply the token information.

In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter by specifying the request as follows:

```
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current?
token=$(puppet-access show)"

curl --insecure "$uri"
```



CAUTION: If you are using curl commands with the `--k` or `--insecure` SSL connection options, keep in mind that you are vulnerable to a person-in-the-middle attack.

Generate a token for use by a service

If you need to generate a token for use by a service and the token doesn't need to be saved, use the `--print` option.

Before you begin

Install the PE client tools package and configure `puppet-access`.

Run `puppet-access login [username] --print`.

This command generates a token, and then displays the token content as stdout (standard output) rather than saving it to disk.

Tip: When generating a token for a service, consider specifying a longer token lifetime so that you don't have to regenerate the token too frequently.

View token activity

Token activity is logged by the activity service. You can see recent token activity on any user's account in the console.

1. In the console, on the **Access control** page, click the **Users** tab and select the full name of the user you are interested in.
2. Click the **Activity** tab.

Change the token's default lifetime

Tokens have a default authentication lifetime of one hour, but this default value can be adjusted in the console. You can also change a token's maximum authentication lifetime, which defaults to 10 years.

1. In the console, click **Node groups**.
2. Open the **PE Infrastructure** node group and click the **PE Console** node group.
3. On the **Classes** tab, find the `puppet_enterprise::profile::console` class.
4. In the **Parameter** field, select `rbac_token_auth_lifetime` to change the default lifetime of all tokens, or `rbac_token_maximum_lifetime` to adjust the maximum allowable lifetime for all tokens.
5. In the **Value** field, enter the new default authentication lifetime.

Specify a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). For example, "12h" generates a token valid for 12 hours.

Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds.

The `rbac_token_auth_lifetime` cannot exceed the `rbac_token_maximum_lifetime` value.

6. Click **Add parameter**, and commit changes.

Set a token-specific lifetime

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

Specify a numeric value followed by "y" (years), "d" (days), "h" (hours), "m" (minutes), or "s" (seconds). For example, "12h" generates a token valid for 12 hours.

Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds.

Use the `--lifetime` parameter if using `puppet-access` to generate your token. For example: `puppet-access login --lifetime 1h`.

Use the `lifetime` value if using the RBAC v1 API to generate your token. For example: `{"login": "<YOUR PE USER NAME>", "password": "<YOUR PE PASSWORD>", "lifetime": "1h"}`.

Set a token-specific label

You can affix a plain-text, user-specific label to tokens you generate with the RBAC v1 API. Token labels allow you to more readily refer to your token when working with RBAC API endpoints, or when revoking your own token.

Token labels are assigned on a per-user basis: two users can each have a token labelled “my token”, but a single user cannot have two tokens both labelled “my token.” You cannot use labels to refer to other users’ tokens.

Generate a token using the `token` endpoint of the RBAC API, using the `label` value to specify the name of your token.

```
{ "login": "<YOUR PE USER NAME>", "password": "<YOUR PE PASSWORD>", "label": "Ava's token" }
```

Labels must be no longer than 200 characters, must not contain commas, and must contain something besides whitespace.

Revoke a token using the API

Revoke tokens by username, label, or full token through the `token` endpoint of the [v2 RBAC API](#). All token revocation attempts are logged in the activity service, and can be viewed on the user’s **Activity** tab in the console.

You can revoke your own token by username, label, or full token. You can also revoke any other full token you possess. Users with the permission to revoke other users can also revoke those users’ tokens, as the `users:disable` permission includes token revocation. Revoking a user’s tokens does not revoke the user’s PE account.

If a user’s account is revoked, all tokens associated with that user account are also automatically revoked.

Revoke a token in the console

Revoke your tokens on the **My Account** page in the console. Administrators can also revoke tokens of other users.

To revoke your token:

1. In the console, on the **My account** page, click the **Tokens** tab.
2. Find the token you want to revoke and click **Revoke token**.

Administrators can revoke another user’s token on the **User details** page.

Delete a token file

If you logged into `puppet-access` to generate a token, you can remove the file that stores that token simply by running the `delete-token-file` command. This is useful if you are working on a server that is used by other people.

Deleting the token file prevents other users from using your authentication token, but does not actually revoke the token. After the token has expired, there’s no risk of obtaining the contents of the token file.

From the command line, run one of the following commands, depending on the path to your token file:

- If your token is at the default token file location, run `puppet-access delete-token-file`.
- If you used a different path to store your token file, run `puppet-access delete-token-file --token-path <YOUR TOKEN PATH>`.

RBAC API v1

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

- [Endpoints](#) on page 224

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

- [Forming RBAC API requests](#) on page 225

Token-based authentication is required to access the RBAC API. You can authenticate requests by using either user authentication tokens or allowed certificates.

- [Users endpoints](#) on page 226

RBAC enables you to manage local users as well as those who are created remotely, on a directory service. With the `users` endpoints, you can get lists of users and create new local users.

- [User group endpoints](#) on page 231

Groups are used to assign roles to a group of users, which is vastly more efficient than managing roles for each user individually. The `groups` endpoints enable you to get lists of groups, and to add a new directory group.

- [User roles endpoints](#) on page 234

By assigning roles to users, you can manage them in sets that are granted access permissions to various PE objects. This makes tracking user access more organized and easier to manage. The `roles` endpoints enable you to get lists of roles and create new roles.

- [Permissions endpoints](#) on page 237

You assign permissions to user roles to manage user access to objects. The `permissions` endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

- [Token endpoints](#) on page 239

A user's access to PE services can be controlled using authentication tokens. Users can generate their own authentication tokens using the `token` endpoint.

- [LDAP endpoints](#) on page 242

Use the LDAP `ds` (directory service) API endpoints to get information about the LDAP directory service, test your LDAP directory service connection, and replace LDAP directory service connection settings.

- [SAML endpoints](#) on page 244

- [Password endpoints](#) on page 246

When local users forget passwords or lock themselves out of PE by attempting to log in with incorrect credentials too many times, you must generate a password reset token for them. The `password` endpoints enable you to generate password reset tokens for a specific local user or with a token that contains a temporary password in the body.

- [RBAC service errors](#) on page 247

You're likely to encounter some errors when using the RBAC API. You'll want to familiarize yourself with the error response descriptions and the general error responses.

- [Configuration options](#) on page 250

There are various configuration options for the RBAC service. Each section can exist in its own file or in separate files.

Related information

[API index](#) on page 26

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

Endpoints

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

For general information about forming HTTP requests to the API, see the [forming requests](#) page. For information on errors encountered while using the RBAC v1 API, see the [RBAC service errors](#) page.

Tip: In addition to the endpoints on this page and in the v2 RBAC service API, there are endpoints that you can use to check the health of the RBAC service. These are available through the [status API](#) documentation.

Endpoint	Use
<code>users</code>	Manage local users as well as those from a directory service, get lists of users, and create new local users.
<code>groups</code>	Get lists of groups and add a new remote user group.

Endpoint	Use
roles	Get lists of user roles and create new roles.
permissions	Get information about available objects and the permissions that can be constructed for those objects.
ds (Directory service)	Get information about the directory service, test your directory service connection, and replace directory service connection settings.
password	Generate password reset tokens and update user passwords.
token	Generate the authentication tokens used to access PE.
rbac-service	Check the status of the RBAC service.

Forming RBAC API requests

Token-based authentication is required to access the RBAC API. You can authenticate requests by using either user authentication tokens or allowed certificates.

By default, the RBAC service listens on port 4433. All endpoints are relative to the `/rbac-api/` path. So, for example, the full URL for the `/v1/users` endpoint on localhost is `https://localhost:4433/rbac-api/v1/users`.

Authentication using tokens

Insert a user authentication token in an RBAC API request.

1. Generate a token: `puppet-access login`
2. Print the token and copy it: `puppet-access show`
3. Save the token as an environment variable: `export TOKEN=<PASTE THE TOKEN HERE>`
4. Include the token variable in your API request:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current"

curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The example above uses the X-Authentication header to supply the token information. In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter. To supply the token in a token parameter, specify the request as follows:

```
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current?
token=$(puppet-access show)"

curl --insecure "$uri"
```



CAUTION: Be aware when using the token parameter method that the token parameter might be recorded in server access logs.

Authenticating using an allowed certificate

You can also authenticate requests using a certificate listed in RBAC's certificate allowlist, located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. Note that if you edit this file, you must reload the `pe-console-services` service (run `sudo service pe-console-services reload`) for your changes to take effect.

Attach the certificate using the command line, as demonstrated in the example curl query below. You must have the allowed certificate name (which must match a name in the `/etc/puppetlabs/console-services/rbac-certificate-allowlist` file) and the private key to run the script.

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current"

curl --cert "$cert" --cacert "$cacert" --key "$key" "$uri"
```

You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate specifically for use with the API.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Content-type headers in the RBAC API

RBAC accepts only JSON payloads in PUT and POST requests.

If a payload is provided, it is important to specify that the content is in JSON format. Thus, all PUT and POST requests with non-empty bodies should have the `Content-Type` header set to `application/json`.

Users endpoints

RBAC enables you to manage local users as well as those who are created remotely, on a directory service. With the users endpoints, you can get lists of users and create new local users.

Users keys

The following keys are used with the RBAC v1 API's users endpoints.

Key	Explanation	Example
id	A UUID string identifying the user.	"4fee7450-54c7-11e4-916c-0800200c9a"
login	A string used by the user to log in. Must be unique among users and groups.	"admin"
email	An email address string. Not currently utilized by any code in PE.	"hill@example.com"
display_name	The user's name as a string.	"Kalo Hill"
role_ids	An array of role IDs indicating which roles should be directly assigned to the user. An empty array is valid.	[3 6 5]
is_group	These flags indicate the type of user. <code>is_group</code> should always be false for a user.	true/false
is_remote		
is_superuser		
is_revoked	Setting this flag to <code>true</code> prevents the user from accessing any routes until the flag is unset or the user's password is reset via token.	true/false

Key	Explanation	Example
last_login	This is a timestamp in UTC-based ISO-8601 format (YYYY-MM-DDThh:mm:ssZ) indicating when the user last logged in. If the user has never logged in, this value is null.	"2014-05-04T02:32:00Z"
inherited_role_ids (remote users only)	An array of role IDs indicating which roles a remote user inherits from their groups.	[9 1 3]
group_ids (remote users only)	An array of UUIDs indicating which groups a remote user inherits roles from.	["3a96d280-54c9-11e4-916c-0800200c9a66"]

GET /users

Fetches all users, both local and remote (including the superuser). Supports filtering by ID through query parameters. Authentication is required.

Request format

To request all the users:

```
GET /users
```

To request specific users, add a comma-separated list of user IDs:

```
GET /users?
id=fe62d770-5886-11e4-8ed6-0800200c9a66,1cadd0e0-5887-11e4-8ed6-0800200c9a66
```

Response format

The response is a JSON object that lists the metadata for all requested users.

For example:

```
[{
  "id": "fe62d770-5886-11e4-8ed6-0800200c9a66",
  "login": "Kalo",
  "email": "kalohill@example.com",
  "display_name": "Kalo Hill",
  "role_ids": [1,2,3...],
  "is_group": false,
  "is_remote": false,
  "is_superuser": true,
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z"
},{
  "id": "07d9c8e0-5887-11e4-8ed6-0800200c9a66",
  "login": "Jean",
  "email": "jeanjackson@example.com",
  "display_name": "Jean Jackson",
  "role_ids": [2, 3],
  "inherited_role_ids": [5],
  "is_group": false,
  "is_remote": true,
  "is_superuser": false,
  "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z"
}]
```

```
{, {
  "id": "1cadd0e0-5887-11e4-8ed6-0800200c9a66",
  "login": "Amari",
  "email": "amariperez@example.com",
  "display_name": "Amari Perez",
  "role_ids": [2, 3],
  "inherited_role_ids": [5],
  "is_group" : false,
  "is_remote" : true,
  "is_superuser" : false,
  "group_ids": [ "2ca57e30-5887-11e4-8ed6-0800200c9a66" ],
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z"
}]
```

GET /users/<sid>

Fetches a single user by its subject ID (sid). Authentication is required.

Response format

For all users, the user contains an ID, a login, an email, a display name, a list of role-ids the user is directly assigned to, and the last login time in UTC-based ISO-8601 format (YYYY-MM-DDThh:mm:ssZ), or null if the user has not logged in yet. It also contains an "is_revoked" field, which, when set to true, prevents a user from authenticating.

For example:

```
{ "id": "fe62d770-5886-11e4-8ed6-0800200c9a66",
  "login": "Amari",
  "email": "amariperez@example.com",
  "display_name": "Amari Perez",
  "role_ids": [1,2,3...],
  "is_group" : false,
  "is_remote" : false,
  "is_superuser" : false,
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z" }
```

For remote users, the response additionally contains a field indicating the IDs of the groups the user inherits roles from and the list of inherited role IDs.

For example:

```
{ "id": "07d9c8e0-5887-11e4-8ed6-0800200c9a66",
  "login": "Jean",
  "email": "jeanjackson@example.com",
  "display_name": "Jean Jackson",
  "role_ids": [2,3...],
  "inherited_role_ids": [],
  "is_group" : false,
  "is_remote" : true,
  "is_superuser" : false,
  "group_ids": [ "b28b8790-5889-11e4-8ed6-0800200c9a66" ],
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z" }
```

GET /users/current

Fetches the data about the current authenticated user, with the exact same behavior as GET /users/<sid>, except that <sid> is assumed from the authentication context. Authentication is required.

GET /users/<user-id>/tokens

Fetches a list of tokens for a given user and produces output in JSON format. Authentication required.

Request format

The request accepts the following parameters:

Parameter	Definition
limit	Maximum number of records that can be returned. Defaults to return all records. Optional integer.
offset	Determines the first record to be returned. Defaults to 0. Optional integer.
order_by	Determines the order by which records are returned. Is one of either creation_date, expiration_date, last_active_date, or client. Defaults to creation_date. Optional string.
order	Determines the sort order by specifying either asc or desc. Defaults to asc. Optional string.

Response format

Here is an example response format:

```
{
  "items": [
    {
      "id": <token id>,
      "creation_date": <ISO-8601>,
      "expiration_date": <ISO-8601>,
      "last_active_date": <ISO-8601>,
      "client": <client field>,
      "description": <description field>,
      "session_timeout": <number, present with timeout in minutes if session based token>,
      "label": <optional user label for token (unused feature for the most part)>
    }, ...
  ],
  "pagination": {
    "limit": <present with value if limit applied>,
    "offset": <offset from start of collection>,
    "order_by": <one of "creation_date", "expiration_date", "last_active_date", "client">,
    "order": <one of "asc", "desc">,
    "total": <total number of records including limit and offset>
  }
}
```

Error response

If a user with the provided identifier doesn't exist, a 404 Not Found response is returned.

POST /users

Creates a new local user. You can add the new user to user roles by specifying an array of roles in `role_ids`. You can set a password for the user in `password`. For the password to work in the PE console, it needs to be a minimum of six characters. Authentication is required.

Request format

Accepts a JSON body containing entries for `email`, `display_name`, `login`, `role_ids`, and `password`. The `password` field is optional. The created account is not useful until the password is set.

For example:

```
{ "login": "Kalo",
  "email": "kalohill@example.com",
  "display_name": "Kalo Hill",
  "role_ids": [1,2,3],
  "password": "yabbadabba" }
```

Response format

If the create operation is successful, a 201 Created response with a location header that points to the new resource is returned.

Error responses

If the email or login for the user conflicts with an existing user's login, a 409 Conflict response is returned.

PUT /users/<sid>

Replaces the user with the specified ID (sid) with a new user object. Authentication is required.

Request format

Accepts an updated user object with all keys provided when the object is received from the API. The behavior varies based on user type. All types have a `role_id` array that indicates all the roles the user should belong to. An empty roles array removes all roles directly assigned to the group.

The below examples show what keys must be submitted. Keys marked with asterisks are the only ones that can be changed via the API.

An example for a local user:

```
{ "id": "c8b2c380-5889-11e4-8ed6-0800200c9a66",
  "login": "Amar*",
  "email": "amariperez@example.com",
  "display_name": "Amar*",
  "role_ids": [1, 2, 3],
  "is_group": false,
  "is_remote": false,
  "is_superuser": false,
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z" }
```

An example for a remote user:

```
{ "id": "3271fde0-588a-11e4-8ed6-0800200c9a66",
  "login": "Jean*",
  "email": "jeanjackson@example.com",
  "display_name": "Jean*",
  "role_ids": [4, 1],
  "inherited_role_ids": [],
  "group_ids": [],
```

```
"is_group" : false,
"is_remote" : true,
"is_superuser" : false,
"is_revoked" : false,
"last_login" : "2014-05-04T02:32:00Z" }
```

Response format

The request returns a 200 OK response, along with the user object with the changes made.

Error responses

If the login for the user conflicts with an existing user login, a 409 Conflict response is returned.

DELETE /users/<sid>

Deletes the user with the specified ID (sid), regardless of whether they are a user defined in RBAC or a user defined by a directory service. In the case of directory service users, while this action removes a user from the console, that user is still able to log in (at which point they are re-added to the console) if they are not revoked. Authentication is required.

Remember: The admin user and the api_user cannot be deleted.

Request format

For example, to delete a user with the ID 3982a629-1278-4e38-883e-12a7cac91535 by using a curl command:

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/rbac-api/v1/
users/3982a629-1278-4e38-883e-12a7cac91535"

curl --cert "$cert" --cacert "$cacert" --key "$key" --request DELETE "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response format

If the user is successfully deleted, a 204 No Content response with an empty body is returned.

Error responses

If the current user does not have the users:edit permission for this user group, a 403 Forbidden response is returned.

If a user with the provided identifier does not exist, a 404 Not Found response is returned.

User group endpoints

Groups are used to assign roles to a group of users, which is vastly more efficient than managing roles for each user individually. The groups endpoints enable you to get lists of groups, and to add a new directory group.

Remember: Group membership is determined by the directory service hierarchy and as such, local users cannot be in directory groups.

User group keys

The following keys are used with the RBAC v1 API's groups endpoints.

Key	Explanation	Example
id	A UUID string identifying the group.	"c099d420-5557-11e4-916c-0800200c9a"

Key	Explanation	Example
login	the identifier for the user group on the directory server.	"poets"
display_name	The group's name as a string.	"Poets"
role_ids	An array of role IDs indicating which roles should be inherited by the group's members. An empty array is valid. This is the only field that can be updated via RBAC; the rest are immutable or synced from the directory service.	[3 6 5]
is_group	These flags indicate that the group is a group.	true, true, false, respectively
is_remote		
is_superuser		
is_revoked	Setting this flag to true currently does nothing for a group.	true/false
user_ids	An array of UUIDs indicating which users inherit roles from this group.	["3a96d280-54c9-11e4-916c-0800200c9a66"]

GET /groups

Fetches all groups. Supports filtering by ID through query parameters. Authentication is required.

Request format

The following requests all the groups:

```
GET /groups
```

To request only some groups, add a comma-separated list of group IDs:

```
GET /groups?
id=65a068a0-588a-11e4-8ed6-0800200c9a66,75370a30-588a-11e4-8ed6-0800200c9a66
```

Response format

The response is a JSON object that lists the metadata for all requested groups.

For example:

```
[{
  "id": "65a068a0-588a-11e4-8ed6-0800200c9a66",
  "login": "hamsters",
  "display_name": "Hamster club",
  "role_ids": [2, 3],
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids": ["07d9c8e0-5887-11e4-8ed6-0800200c9a66"]}
],{
  "id": "75370a30-588a-11e4-8ed6-0800200c9a66",
  "login": "chinchilla",
  "display_name": "Chinchilla club",
  "role_ids": [2, 1],
```



```

    "is_group" : true,
    "is_remote" : true,
    "is_superuser" : false,
    "user_ids":
    [ "1cadd0e0-5887-11e4-8ed6-0800200c9a66", "5c1ab4b0-588b-11e4-8ed6-0800200c9a66" ]
  }, {
    "id": "ccdbde50-588a-11e4-8ed6-0800200c9a66",
    "login": "wombats",
    "display_name": "Wombat club",
    "role_ids": [2, 3],
    "is_group" : true,
    "is_remote" : true,
    "is_superuser" : false,
    "user_ids": []
  }
]
```

GET /groups/<sid>

Fetches a single group by its subject ID (sid). Authentication is required.

Response format

The response contains an ID for the group and a list of `role_ids` the group is directly assigned to.

For directory groups, the response contains the display name, the login field, a list of `role_ids` directly assigned to the group, and `user_ids` containing IDs of the remote users that belong to that group.

For example:

```

{ "id": "65a068a0-588a-11e4-8ed6-0800200c9a66",
  "login": "hamsters",
  "display_name": "Hamster club",
  "role_ids": [2, 3],
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids": [ "07d9c8e0-5887-11e4-8ed6-0800200c9a66" ] }
```

Error responses

If the user who submits the GET request has not successfully authenticated, a 401 Unauthorized response is returned.

If the current user does not have the appropriate user permissions to request the group data, a 403 Forbidden response is returned.

POST /groups

Creates a new remote group and attaches to the new group any roles specified in its roles list. Authentication is required.

Request format

Accepts a JSON body containing an entry for `login`, and an array of `role_ids` to assign to the group initially.

For example:

```

{ "login": "Augmentators",
  "role_ids": [1,2,3] }
```

Response format

If the create operation is successful, a 201 Created response with a location header that points to the new resource is returned.

Error responses

If the login for the group conflicts with an existing group login, a 409 Conflict response is returned.

PUT /groups/<sid>

Replaces the group with the specified ID (sid) with a new group object. Authentication is required.

Request format

Accepts an updated group object containing all the keys that were received from the API initially. The only updatable field is `role_ids`. An empty roles array indicates a desire to remove the group from all the roles it was directly assigned to. Any other changed values are ignored.

For example:

```
{ "id": "75370a30-588a-11e4-8ed6-0800200c9a66",
  "login": "chinchilla",
  "display_name": "Chinchillas",
  **"role_ids": [2, 1]**,
  "is_group" : true,
  "is_remote" : true,
  "is_superuser" : false,
  "user_ids":
    [ "1cadd0e0-5887-11e4-8ed6-0800200c9a66", "5c1ab4b0-588b-11e4-8ed6-0800200c9a66" ] }
```

Response format

If the operation is successful, a 200 OK response including a group object with updated roles is returned.

DELETE /groups/<sid>

Deletes the user group with the specified ID (sid) from RBAC without making any changes to the directory service. Authentication required.

Response format

If the user group is successfully deleted, a 204 No Content with an empty body is returned.

Error responses

If the current user does not have the `user_groups:delete` permission for this user group, a 403 Forbidden response is returned.

If a group with the provided identifier does not exist, a 404 Not Found response is returned.

User roles endpoints

By assigning roles to users, you can manage them in sets that are granted access permissions to various PE objects. This makes tracking user access more organized and easier to manage. The `roles` endpoints enable you to get lists of roles and create new roles.

User role keys

The following keys are used with the RBAC v1 API's roles endpoints.

Key	Explanation	Example
<code>id</code>	An integer identifying the role.	18
<code>display_name</code>	The role's name as a string.	"Viewers"
<code>description</code>	A string describing the role's function.	"View-only permissions"

Key	Explanation	Example
permissions	An array containing permission objects that indicate what permissions a role grants. An empty array is valid. See Permission keys for more information.	[]
user_ids	An array of UUIDs indicating which users and groups are directly assigned to the role. An empty array is valid.	["fc115750-555a-11e4-916c-0800200c9a66", ...]
group_ids		

GET /roles

Fetches all roles with user and group ID lists and permission lists. Authentication is required.

Response format

Returns a JSON object containing all roles with user and group ID lists and permission lists.

For example:

```
[{"id": 123,
  "permissions": [{"object_type": "node_groups",
                    "action": "edit_rules",
                    "instance": "*"}, ...],
  "user_ids": [
    "1cadd0e0-5887-11e4-8ed6-0800200c9a66", "5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],
  "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
  "display_name": "A role",
  "description": "Edit node group rules"},
...]
```

GET /roles/<rid>

Fetches a single role by its ID (rid). Authentication is required.

Response format

Returns a 200 OK response with the role object with a full list of permissions and user and group IDs.

For example:

```
{"id": 123,
  "permissions": [{"object_type": "node_groups",
                    "action": "edit_rules",
                    "instance": "*"}, ...],
  "user_ids": [
    "1cadd0e0-5887-11e4-8ed6-0800200c9a66", "5c1ab4b0-588b-11e4-8ed6-0800200c9a66"],
  "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"],
  "display_name": "A role",
  "description": "Edit node group rules"}
```

POST /roles

Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

Permissions keys for task-targets

If you're writing a role for a task-target, you must include unique `action` and `instance` key values to specify permissions. For the complete task-target workflow, see the blog post [Puppet Enterprise RBAC API, or how to manage access to tasks](#).

Key	Value	Explanation
<code>action</code>	<code>run_with_constraints</code>	Specifies that the user has permission to run a task on certain nodes within the confines of a given task-target.
<code>instance</code>	<code><task-target ID></code>	Specifies the ID of the task-target the user has permission to run.

Request format

Accepts a new role object. Any of the arrays can be empty and "description" can be null.

For example:

```
{
  "permissions": [
    {
      "object_type": "node_groups",
      "action": "edit_rules",
      "instance": "*"
    },
    ...
  ],
  "user_ids": [
    "1cadd0e0-5887-11e4-8ed6-0800200c9a66",
    "5c1ab4b0-588b-11e4-8ed6-0800200c9a66"
  ],
  "group_ids": [
    "2ca57e30-5887-11e4-8ed6-0800200c9a66"
  ],
  "display_name": "A role",
  "description": "Edit node group rules"
}
```

Response format

Returns a 201 Created response with a location header pointing to the new resource.

Error responses

Returns a 409 Conflict response if the role has a name that collides with an existing role.

Related information

[POST /command/task_target](#) on page 561

Create a new task-target, which is a collection of tasks, nodes and node groups that define a permission group. When a user has permissions to run via a given task-target, they are granted permissions to run the given set of tasks, on the set of nodes described by the task-target.

PUT /roles/<rid>

Replaces a role at the specified ID (rid) with a new role object. Authentication is required.

Request format

Accepts the modified role object.

For example:

```
{
  "id": 123,
  "permissions": [
    {
      "object_type": "node_groups",
      "action": "edit_rules",
      "instance": "*"
    },
    ...
  ],
}
```

```
"user_ids":
[ "1cadd0e0-5887-11e4-8ed6-0800200c9a66", "5c1ab4b0-588b-11e4-8ed6-0800200c9a66" ],
"group_ids": [ "2ca57e30-5887-11e4-8ed6-0800200c9a66" ],
"display_name": "A role",
"description": "Edit node group rules"}
```

Response format

Returns a 200 OK response with the modified role object.

Error responses

Returns a 409 Conflict response if the new role has a name that collides with an existing role.

DELETE /roles/<rid>

Deletes the role identified by the role ID (rid). Users with this role immediately lose the role and all permissions granted by it, but their session is otherwise unaffected. Access to the next request that the user makes is determined by the new set of permissions the user has without this role.

Response format

Returns a 200 OK response if the role identified by <rid> has been deleted.

Error responses

Returns a 404 Not Found response if no role exists for <rid>.

Returns a 403 Forbidden response if the current user lacks permission to delete the role identified by <rid>.

Permissions endpoints

You assign permissions to user roles to manage user access to objects. The permissions endpoints enable you to get information about available objects and the permissions that can be constructed for those objects.

Permissions keys

The following keys are used with the RBAC v1 API's permissions endpoints. The available values for these keys are available from the /types endpoint (see below).

Key	Explanation	Example
object_type	A string identifying the type of object this permission applies to.	"node_groups"
action	A string indicating the type of action this permission permits.	"modify_children"
instance	A string containing the primary ID of the object instance this permission applies to, or "*" indicating that it applies to all instances. If the given action does not allow instance specification, "*" should always be used.	"cec7e830-555b-11e4-916c-0800200c9a66" or "*"

GET /types

Lists the objects that integrate with RBAC and demonstrates the permissions that can be constructed by picking the appropriate object_type, action, and instance triple. Authentication is required.

The has_instances flag indicates that the action permission is instance-specific if true, or false if this action permission does not require instance specification.

Response format

Returns a 200 OK response with a listing of types.

For example:

```
[{ "object_type": "node_groups",
  "display_name": "Node Groups",
  "description": "Groups that nodes can be assigned to.",
  "actions": [{ "name": "view",
                 "display_name": "View",
                 "description": "View the node groups",
                 "has_instances": true
               }, {
                 "name": "modify",
                 "display_name": "Configure",
                 "description": "Modify description, variables and classes",
                 "has_instances": true
               }, ...]
}, ...]
```

Error responses

Returns a 401 Unauthorized response if no user is logged in.

Returns a 403 Forbidden response if the current user lacks permissions to view the types.

POST /permitted

Checks an array of permissions for the subject identified by the submitted identifier.

Request format

This endpoint takes a "token" in the form of a user or a user group's UUID and a list of permissions. This returns true or false for each permission queried, representing whether the subject is permitted to take the given action.

The full evaluation of permissions is taken into account, including inherited roles and matching general permissions against more specific queries. For example, a query for `users:edit:1` returns true if the subject has `users:edit:1` or `users:edit:*`.

In the following example, the first permission is querying whether the subject specified by the token is permitted to perform the `edit_rules` action on the instance of `node_groups` identified by the ID 4. Note that in reality, node groups and users use UUIDs as their IDs.

```
{ "token": "<subject uuid>",
  "permissions": [{ "object_type": "node_groups",
                    "action": "edit_rules",
                    "instance": "4" },
                  { "object_type": "users",
                    "action": "disable",
                    "instance": "1" } ]
}
```

Response format

Returns a 200 OK response with an array of Boolean values representing whether each submitted action on a specific object type and instance is permitted for the subject. The array always has the same length as the submitted array and each returned Boolean value corresponds to the submitted permission query at the same index.

The example response below was returned from the example request in the previous section. This return means the subject is permitted `node_groups:edit_rules:4` but not permitted `users:disable:1`.

```
[true, false]
```

GET /permitted/<object-type>/<action>

Return an array of permitted instances for the given `object-type/action` pair based on the current user authentication context.

This endpoint returns the instance values as an array for the given `object-type` and `action` pair. If the user does not have permissions for any instance, an empty array will be returned.

If the `object-type` does not map to a known `object-type`, the endpoint will return a 404.

If the `action` does not exist for the given `object-type`, the endpoint will return a 404.

Example

GET /permitted/node_groups/view

```
[ "00000000-0000-4000-8000-000000000000" ]
```

Returns a 200 OK response with a listing of instances.

GET /permitted/<object-type>/<action>/<uuid>

Return an array of permitted instances for the given `object-type/action` pair based on the user matching the supplied `uuid`.

This endpoint returns the instance values as an array for the given `object-type` and `action` pair. If the user does not have permissions for any instances, an empty array will be returned.

If the `object-type` does not map to a known `object-type`, the endpoint will return a 404.

If the `action` does not exist for the given `object-type`, the endpoint will return a 404.

If the `uuid` does not map to a known user, the endpoint will return a 404.

Example

GET /permitted/node_groups/view/fe62d770-5886-11e4-8ed6-0800200c9a66

```
[ "00000000-0000-4000-8000-000000000000" ]
```

Returns a 200 OK response with a listing of instances.

Token endpoints

A user's access to PE services can be controlled using authentication tokens. Users can generate their own authentication tokens using the token endpoint.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Token keys

The following keys are used with the token endpoint.

Key	Explanation
login	The user's login for the PE console (required).

Key	Explanation
password	The user's password for the PE console (required).
lifetime	The length of time the token is active before expiration (optional).
description	Additional metadata about the requested token (optional).
client	Additional metadata about the client making the token request (optional).
label	A user-defined label for the token (optional).

The lifetime key

When setting a token's lifetime, specify a numeric value followed by *y* (years), *d* (days), *h* (hours), *m* (minutes), or *s* (seconds). For example, a value of 12h is 12 hours. Do not add a space between the numeric value and the unit of measurement. If you do not specify a unit, it is assumed to be seconds. If you do not want the token to expire, set the lifetime to 0. Setting it to zero gives the token a lifetime of approximately 10 years.

Tip: The default lifetime for all tokens is also configurable. See [Change the token's default lifetime](#) for configuration instructions.

The label key

You can choose to select a label for the token that can be used with other RBAC token endpoints. Labels:

- Must be no longer than 200 characters.
- Must not contain commas.
- Must contain something other than whitespace. (Whitespace is trimmed from the beginning and end of the label, though it is allowed elsewhere.)
- Must not be the same as a label for another token for the same user.

Token labels are assigned on a per-user basis: two users can each have a token labelled `my token`, but a single user cannot have two tokens both labelled `my token`. You cannot use labels to refer to other users' tokens.

POST /auth/token

Generates an access token for the user whose login information is POSTed. This token can then be used to authenticate requests to PE services using either the `X-Authentication` header or the `token` query parameter.

This route is intended to require zero authentication. While HTTPS is still required (unless PE is explicitly configured to permit HTTP), neither an allowed cert nor a session cookie is needed to POST to this endpoint.

Request format

Accepts a JSON object or curl command with the user's login and password information. The token's lifetime, a user-specified label, and additional metadata can be added, but are not required.

An example JSON request:

```
{ "login": "jeanjackson@example.com",
  "password": "1234",
  "lifetime": "4m",
  "label": "personal workstation token" }
```

An example curl command request:

```
type_header='Content-Type: application/json'
cacert="$(puppet config print cacert)"
uri="https://$(puppet config print server):4433/rbac-api/v1/auth/token"
```



```
data='{ "login": "<USER>",
      "password": "<PASSWORD>",
      "lifetime": "4h",
      "label": "four-hour token"}'

curl --header "$type_header" -cacert "$cacert" --request POST "$uri" --data
"$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The various parts of this curl command request are explained as follows:

- `--header 'Content-Type: application/json'`: sets the Content-Type header to `application/json`, which indicates to RBAC that the data being sent is in JSON format.
- `--cacert [FILE]`: Specifies a CA certificate as described in [Forming requests for the node classifier](#). Alternatively, you could use the `--insecure` flag to turn off SSL verification of the RBAC server so that you can use the HTTPS protocol without providing a CA cert. If you do not provide one of these options in your curl request, curl complains about not being able to verify the RBAC server.

Note: The `--insecure` flag is shown as an example only. You should use your own discretion when choosing the appropriate server verification method for the tool that you are using.

- `--request POST`: This is an HTTP POST request to provide your login information to the RBAC service.
- `https://<HOSTNAME>:<PORT>/rbac-api/v1/auth/token`: Sends the request to the token endpoint. For `HOSTNAME`, provide the FQDN of the server that is hosting the PE console service. If you are making the call from the console server, you can use `"localhost."` For `PORT`, provide the port that the PE services (node classifier service, RBAC service, and activity service) listen on. The default port is 4433
- `--data '{"login": "<USER>", "password": "<PASSWORD>", "lifetime": "4h", "label": "four-hour token"}'`: Provide the user name and password that you use to log in to the PE console. Optionally, set the token's lifetime and label.

Response format

Returns a 200 OK response if the credentials are good and the user is not revoked, along with a token.

For example:

```
{ "token": "asd0u0=2jdi jasodj-
w0duwdhjashd,kjsahdasoi0d9hw0hduashd0a9wdy0whdkaudhaksdhc9chakdh92..." }
```

Error responses

Returns a 401 Unauthenticated response if the credentials are bad or the user is revoked.

Returns a 400 Malformed response if something is wrong with the request body.

POST /tokens

Create a new token with a custom lifetime, client specification, and description. Cannot be used with certificate authentication.

Request format

The request accepts a JSON payload with the following keys:

Key	Definition
lifetime	Required string specifying the lifetime of the token in the <code><whole-number></code> form, with one of (s,m,d,y). Use s for seconds, m for minutes, d for days, and y for years. for example, 5m is 5 minutes.

Key	Definition
client	Required string specifying a human readable description of the creator of the token. For example, PE console.
description	Optional string specifying a human readable description of the token.

For example:

```
{
  "lifetime": "1y",
  "description": "A token to be used with joy and care.",
  "client": "PE console"
}
```

Response format

Returns a 200 OK response when the token is valid, not expired, and the user is not revoked.

Returns standard RBAC errors when the user doesn't exist, is revoked, etc.

For example:

```
{
  "token": "ALDbkgFtOANbMtACmD1R7BA42cNLDnN3VthXpBIkMQyg"
}
```

LDAP endpoints

Use the LDAP ds (directory service) API endpoints to get information about the LDAP directory service, test your LDAP directory service connection, and replace LDAP directory service connection settings.

To connect to the directory service anonymously, specify null for the lookup user and lookup password or leave these fields blank.

Related information

[External directory settings](#) on page 205

The table below provides examples of the settings used to connect to an Active Directory service and an OpenLDAP service to PE. Each setting is explained in more detail below the table.

GET /ds

Get the connected directory service information. Authentication is required.

Return format

Returns a 200 OK response with an object representing the connection.

For example:

```
{ "display_name": "AD",
  "hostname": "ds.foobar.com",
  "port": 10379, ... }
```

If the connection settings have not been specified, returns a 200 OK response with an empty JSON object.

For example:

```
{ }
```

GET /ds/test

Runs a connection test for the connected directory service. Authentication is required.

Return format

If the connection test is successful, returns a 200 OK response with information about the test run.

For example:

```
{ "elapsed": 10 }
```

Error responses

- 400 Bad Request if the request is malformed.
- 401 Unauthorized if no user is logged in.
- 403 Forbidden if the current user lacks the permissions to test the directory settings.

```
{ "elapsed": 20, "error": "..."} 
```

PUT /ds/test

Performs a connection test with the submitted settings. Authentication is required.

Request format

Accepts the full set of directory settings keys with values defined.

Return format

If the connection test is successful, returns information about the test run.

For example:

```
{ "elapsed": 10 }
```

Error responses

If the request times out or encounters an error, returns information about the test run.

For example:

```
{ "elapsed": 20, "error": "..."} 
```

PUT /ds

Replaces current directory service connection settings. Authentication is required.

When changing directory service settings, you must specify all of the required directory service settings in the PUT request, including the required settings that are remaining the same. You do not need to specify optional settings unless you are changing them.

Request format

Accepts directory service connection settings. To "disconnect" the DS, PUT either an empty object ("{}") or the settings structure with all values set to null.

For example:

```
{ "hostname": "ds.somehost.com",  
  "name":  
  "port": 10389,
```

```
"login": "frances", ...}
```

Working with nested groups

When authorizing users, the RBAC service has the ability to search nested groups. Nested groups are groups that are members of external directory groups. For example, if your external directory has a "System Administrators" group, and you've given that group the "Superusers" user role in RBAC, then RBAC also searches any groups that are members of the "System Administrators" group and assign the "Superusers" role to users in those member groups.

By default, RBAC does not search nested groups. To enable nested group searches, specify the `search_nested_groups` field and give it a value of `true`.

Note: In PE 2015.3 and earlier versions, RBAC's default was to search nested groups. When **upgrading** from one of these earlier versions, this default setting is preserved and RBAC continues to search nested groups. If you have a large number of nested groups, you might experience a slowdown in performance when users are logging in because RBAC is searching the entire hierarchy of groups. To avoid performance issues, change the `search_nested_groups` field to `false` for a more shallow search in which RBAC searches only the groups it has been configured to use for user roles.

Using StartTLS connections

To use a StartTLS connection, specify `"start_tls": true`. When set to `true`, StartTLS is used to secure your connection to the directory service, and any certificates that you have configured through the DS trust chain setting is used to verify the identity of the directory service. When specifying StartTLS, make sure that you don't also have SSL set to `true`.

Disabling matching rule in chain

When PE detects an Active Directory that supports the `LDAP_MATCHING_RULE_IN_CHAIN` feature, it automatically uses it. Under some specific circumstances, you might need to disable this setting. To disable it, specify `"disable_ldap_matching_rule_in_chain": true` in the PUT request. This is an optional setting.

Return format

Returns a 200 OK response with an object showing the updated connection settings.

For example:

```
{ "hostname": "ds.somehost.com",
  "port": 10389,
  "login": "frances", ... }
```

SAML endpoints

PUT /saml

Configure SAML. When changing SAML settings, you must specify all of the required settings in the PUT request, including the required settings that are remaining the same. You do not need to specify optional settings unless you are changing them.

Request format

Accepts SAML connection settings. See the [SAML configuration reference](#) on page 212 for the complete list of settings.

Response format

Returns a "201" code if the settings are new and includes the new settings in the payload.

Returns a "200 OK" code if the settings have been updated and set.

Returns a "403" code if the user lacks the `directory_service:edit:*` permission.

GET /saml

Retrieves the currently configured SAML configuration.

Response format

Returns a 200 code with the SAML configuration when SAML is configured and the user is authorized. See the [SAML configuration reference](#) on page 212 for the complete list of settings.

Returns a 404 code if the SAML data is not configured.

DELETE /saml

Remove the existing SAML configuration.

Response format

Returns "204 no content" code if the SAML configuration is removed correctly.

Returns a "404 not found" code if the SAML configuration is not set prior to making the request.

Returns a "403" code if the user lacks the `directory_service:edit*` permission.

GET /v1/saml/meta

Retrieve the public SAML certificate and URLs needed to configure an identity provider.

Configure your identity provider with these key values. After it's configured, your identity provider supplies the required values for configuring SAML in PE. This information is also exposed in the console on the **SSO** tab.

Response format

This endpoint returns a "200 Ok" response if the instance is not a replica and the certificate exists. The following keys are returned:

Key	Definition
meta	A URL to the public metadata endpoint for the SAML service provider. Some IdP configurations also require this URL in the "entity-id" and/or "audience restriction" fields
slo	A URL to the public logout service for SAML.
acs	A URL to the public assertion service for SAML.
cert	A string representing the public SAML certificate.

For example,

```
{
  "meta": "https://localhost/saml/v1/meta",
  "acs": "https://localhost/saml/v1/acs",
  "slo": "https://localhost/saml/v1/slo",
  "cert": "-----BEGIN CERTIFICATE-----\nMIIFo ..."
```

Error response

The endpoint returns a "404 not found" error if the SAML key entries are not present in the configuration, or the public key file does not exist.

Response format

Returns a 200 OK response if the token is valid and the password has been successfully changed.

Error responses

Returns a 403 Permission Denied response if the token has already been used or is invalid.

PUT /users/current/password

Changes the password for the current local user. A payload containing the current password must be provided. Authentication is required.

Request format

The current and new passwords must both be included.

For example:

```
{ "current_password": "somepassword",
  "password": "someotherpassword" }
```

Response format

Returns a 200 OK response if the password has been successfully changed.

Error responses

Returns a 403 Forbidden response if the user is a remote user, or if `current_password` doesn't match the current password stored for the user. The body of the response includes a message that specifies the cause of the failure.

RBAC service errors

You're likely to encounter some errors when using the RBAC API. You'll want to familiarize yourself with the error response descriptions and the general error responses.

Error response format

When the client specifies an `accept` header in the request with type `application/json`, the RBAC service returns errors in a standard format.

Each response is an object containing the following keys:

Key	Definition
<code>kind</code>	A string classifying the error. It should be the same for all errors that have the same type of information in their <code>details</code> key.
<code>msg</code>	A human-readable message describing the error.
<code>details</code>	Additional machine-readable information about the error condition. The format of this key's value varies between kinds of errors, but is the same for each kind of error.

When returning errors in `text/html`, the body is the contents of the `msg` field.

General error responses

Any endpoint accepting a JSON body can return several kinds of 400 Bad Request responses.

Response	Status	Description
malformed-request	400	The submitted data is not valid JSON. The <code>details</code> key consists of one field, <code>error</code> , which contains the error message from the JSON parser.
schema-violation	400	<p>The submitted data has an unexpected structure, such as invalid fields or missing required fields. The <code>msg</code> contains a description of the problem. The <code>details</code> are an object with three keys:</p> <ul style="list-style-type: none"> • <code>submitted</code>: The submitted data as it was seen during schema validation. • <code>schema</code>: The expected structure of the data. • <code>error</code>: A structured description of the error.
inconsistent-id	400	Data was submitted to an endpoint where the ID of the object is a part of the URL and the submitted data contains an <code>id</code> field with a different value. The <code>details</code> key consists of two fields, <code>url-id</code> and <code>body-id</code> , showing the IDs from both sources.
invalid-id-filter	400	A URL contains a filter on the ID with an invalid format. No details are given with this error.
invalid-uuid	400	An invalid UUID was submitted. No details are given with this error.
user-unauthenticated	401	An unauthenticated user attempted to access a route that requires authentication.
user-revoked	401	A user who has been revoked attempted to access a route that requires authentication.
api-user-login	401	A person attempted to log in as the <code>api_user</code> with a password (<code>api_user</code> does not support username/password authentication).

Response	Status	Description
remote-user-conflict	401	<p>A remote user who is not yet known to RBAC attempted to authenticate, but a local user with that login already exists.</p> <p>The solution is to change either the local user's login in RBAC, or to change the remote user's login, either by changing the <code>user_lookup_attr</code> in the DS settings or by changing the value in the directory service itself.</p>
permission-denied	403	A user attempted an action that they are not permitted to perform.
admin-user-immutable	403	A user attempted to edit metadata or associations belonging to the default roles ("Administrators", "Operators", "Code Deployers", or "Viewers") or default users ("admin" or "api_user") that they are not allowed to change.
admin-user-not-in-admin-role		
default-roles-immutable		
conflict	409	A value for a field that is supposed to be unique was submitted to the service and another object has that value. For example, when a user is created with the same login as an existing user.
invalid-associated-id	422	An object was submitted with a list of associated IDs (for example, <code>user_ids</code>) and one or more of those IDs does not correspond to an object of the correct type.
no-such-user-LDAP	422	An object was submitted with a list of associated IDs (for example, <code>user_ids</code>) and one or more of those IDs does not correspond to an object of the correct type.
no-such-group-LDAP		
non-unique-lookup-attr	422	A login was attempted but multiple users are found via LDAP for the given username. The directory service settings must use a <code>user_lookup_attr</code> that is guaranteed to be unique within the provided user's RDN.
server-error	500	Occurs when the server throws an unspecified exception. A message and stack trace should be available in the logs.

Configuration options

There are various configuration options for the RBAC service. Each section can exist in its own file or in separate files.

RBAC service configuration

You can configure the RBAC service's settings to specify the duration before inactive user accounts expire, adjust the length of user sessions, the number of times a user can attempt to log in, and the length of time a password reset token is valid. You can also create a list of trusted, or allowed, certificates.

These configuration parameters are not required, but when present must be under the `rbac` section, as in the following example:

```
rbac: {
  # Duration in days before an inactive account expires
  account-expiry-days: 1
  # Duration in minutes that idle user accounts are checked
  account-expiry-check-minutes: 60
  # Duration in hours that a password reset token is viable
  password-reset-expiration: 24
  # Duration in minutes that a session is viable
  session-timeout: 60
  failed-attempts-lockout: 10
}
```

account-expiry-days

This parameter is a positive integer that specifies the duration, in days, before an inactive user account expires. The default value is undefined. To activate this feature, add a value of 1 or greater.

If a non-superuser hasn't logged into the console during this specified period, their user status updates to revoked. After creating an account, if a non-superuser hasn't logged in to the console during the specified period, their user status updates to revoked.

Important: If the `account-expiry-days` parameter is not specified, or has a value of less than 1, the `account-expiry-check-minutes` parameter is ignored.

account-expiry-check-minutes

This parameter is a positive integer that specifies how often, in minutes, the application checks for idle user accounts. The default value is 60 minutes.

password-reset-expiration

When a user doesn't remember their current password, an administrator can generate a token for them to change their password. The duration, in hours, that this generated token is valid can be changed with the `password-reset-expiration` config parameter. The default value is 24.

session-timeout

This parameter is a positive integer that specifies how long a user's session should last, in minutes. This session is the same across node classifier, RBAC, and the console. The default value is 60.

failed-attempts-lockout

This parameter is a positive integer that specifies how many failed login attempts are allowed on an account before that account is revoked. The default value is 10.

Note: If you change this value, create a new file or Puppet resets back to 10 when it next runs. Create the file in an RBAC section of `/etc/puppetlabs/console-services/conf.d`.

certificate-allowlist

This parameter is a path for specifying the file that contains the names of hosts that are allowed to use RBAC APIs and other downstream component APIs, such as the Node Classifier and the Activity services. This configuration is for the users who want to script interaction with the RBAC service.

Users must connect to the RBAC service with a client certificate that has been specified in this `certificate-allowlist` file. A successful match of the client certificate and a certificate on this list allows access to the RBAC APIs as the `api_user`. By default, this user is an administrator and has all available permissions.

The certificate allowlist contains, at minimum, the certificate for the nodes PE is installed on.

RBAC database configuration

Credential information for the RBAC service is stored in a PostgreSQL database.

The configuration information for that database is found in the 'rbac-database' section of the config.

For example:

```
rbac-database: {
  classname: org.postgresql.Driver
  subprotocol: postgresql
  subname: "//<path-to-host>:5432/perbac"
  user: <username here>
  password: <password here>
}
```

classname

Used by the RBAC service for connecting to the database; this option should always be `org.postgresql.Driver`.

subprotocol

Used by the RBAC service for connecting to the database; this options should always be `postgresql`.

subname

JDBC connection path used by the RBAC service for connecting to the database. This should be set to the hostname and configured port of the PostgreSQL database. `perbac` is the database the RBAC service uses to store credentials.

user

This is the username the RBAC service should use to connect to the PostgreSQL database.

password

This is the password the RBAC service should use to connect to the PostgreSQL database.

RBAC API v2

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

- [User group endpoints](#) on page 252

Groups are used to assign roles to a group of users, which is more efficient than managing roles for each user individually. The `groups` endpoint enables you to create new remote groups.

- [Tokens endpoints](#) on page 253

A user's access to PE services can be controlled using authentication tokens. Users can revoke authentication tokens using the `tokens` endpoints.

Related information

[API index](#) on page 26

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

User group endpoints

Groups are used to assign roles to a group of users, which is more efficient than managing roles for each user individually. The `groups` endpoint enables you to create new remote groups.

POST /groups

Create a new remote group and attach roles to it that are specified in its roles list. Authentication is required.

Request format

The endpoint accepts a JSON body containing the following keys:

Key	Definition
<code>login</code>	Defines the group for an external IdP. This could be an LDAP login or a SAML identifier for the group. Required.
<code>role_ids</code>	An array of role IDs to assign to the group initially. Required.
<code>display_name</code>	The name of the group that displays in the console. If this represents an LDAP group, the LDAP group display name will override it. Optional.
<code>validate</code>	Validate that the group exists on the LDAP server prior to creating it. Defaults to <code>true</code> . If <code>false</code> , the group is not validated against LDAP. Optional.

Example:

```
{
  "login": "augmentators",
  "role_ids": [1,2,3]
  "display_name": "The Augmentors"
}
```

Response format

If the new remote group is created successfully, "303 See Other" with a location header pointing to the new resource is returned.

Error response

"409 Conflict" is returned if the new group conflicts with an existing group.

Tokens endpoints

A user's access to PE services can be controlled using authentication tokens. Users can revoke authentication tokens using the `tokens` endpoints.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

DELETE /tokens

Use this endpoint to revoke one or more authentication tokens, ensuring the tokens can no longer be used with RBAC.

Parameters

The tokens must be specified using at least one of the following parameters:

Parameter	Value
<code>revoke_tokens</code>	A list of complete authentication tokens to be revoked.
<code>revoke_tokens_by_usernames</code>	A list of usernames whose tokens are to be revoked.
<code>revoke_tokens_by_labels</code>	A list of the labels of tokens to revoke (only tokens owned by the user making the request can be revoked in this manner).
<code>revoke_tokens_by_ids</code>	A list of token ids whose tokens are to be revoked.

You can supply parameters either as query parameters or as JSON-encoded in the body. If you include them as query parameters, separate the tokens, labels, or usernames by commas:

```
/tokens?revoke_tokens_by_usernames=<USER NAME>,<USER NAME>
```

If you encoded them in the body, supply them as a JSON array:

```
{ "revoke_tokens_by_usernames": [ "<USER NAME>", "<USER NAME>" ] }
```

If you provide a parameter as both a query parameter and in the JSON body, the values from the two sources are combined. It is not an error to specify the same token using multiple means (such as by providing the entire token to the `revoke_tokens` parameter and also including its label in the value of `revoke_tokens_by_labels`).

In the case of an error, malformed or otherwise input, or bad request data, the endpoint still attempts to revoke as many tokens as possible. This means it's possible to encounter multiple error conditions in a single request, such as if there were malformed usernames supplied in the request *and* a database error occurred when trying to revoke the well-formed usernames.

All operations on this endpoint are idempotent—it is not an error to revoke the same token two or more times.

Any user can revoke any token by supplying the complete token in the `revoke_tokens` query parameter or request body field.

To revoke tokens by username, the user making the request must have the "Users Revoke" permission for that user.

Example JSON body

The following is an example JSON body using each of the available parameters.

```
{ "revoke_tokens" :
  [ "eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhZG1pb2IiImhhdCI6MTQzOTQ5Mzg0NiwiZXh0Ij06ImR0C5yeSBGWfX-0KEbp42cGz8lA6rrIHpsaajRWUg9yThEUKT2crh6878orCLgfoBLDh-roBTLeIual6sasH-ggpDhQVktFOomEXM6UTJlp1NpuP01rNr9Jm1xWhI8WpExH1l_-136D1NJm32kwo-oV6GzXR70xq_N2CwIwObw-X1S5aUUC4KkyPtDmNvnvCln4" ]
  "revoke_tokens_by_labels": [ "Workstation Token", "VPS Token" ],
  "revoke_tokens_by_usernames": [ "<USER NAME>", "<USER NAME>" ] }
```

Response codes

The server uses the following response codes:

Code	Definition
204 No Content	Sent if all operations are successful.
500 Application Error	Sent if there was a database error when trying to revoke tokens.
403	Sent if the user lacks the permission to revoke one of the supplied usernames and no database error occurred.
400 Malformed	<p>Sent if one these conditions are true:</p> <ul style="list-style-type: none"> At least one of the tokens, usernames, or labels is malformed. At least one of the usernames does not exist in the RBAC database. None of <code>revoke_tokens</code>, <code>revoke_tokens_by_labels</code>, <code>revoke_tokens_by_usernames</code>, or <code>revoke_tokens_by_ids</code> is supplied. There are unrecognized query parameters or fields in the request body, and the user has all necessary permissions and no database error occurred.

All error responses follow the standard JSON error format, meaning they have `kind`, `msg`, and `details` keys.

The `kind` key is `puppetlabs.rbac/database-token-error` if the response is a 500, `permission-denied` if the response is a 403, and `malformed-token-request` if the response is a 400.

The `msg` key contains an English-language description of the problems encountered while processing the request and performing the revocations, ending with either "No tokens were revoked" or "All other tokens were successfully revoked", depending on whether any operations were successful.

The `details` key contains an object with arrays in the `malformed_tokens`, `malformed_usernames`, `malformed_labels`, `nonexistent_usernames`, `permission_denied_usernames`, `unrecognized_parameters`, `malformed_ids`, and `permission_denied_ids` fields, as well as the Boolean field `other_tokens_revoked`.

The arrays all contain bad input from the request, and the `other_tokens_revoked` field's value indicates whether any of the revocations specified in the request were successful or not.

Example error body

The server returns an error body resembling the following:

```
{
  "kind": "malformed-request",
  "msg": "The following user does not exist: FormerEmployee. All other tokens were successfully revoked.",
  "details": {
    "malformed_tokens": [],
    "malformed_labels": [],
    "malformed_usernames": [],
    "malformed_ids": [],
    "nonexistent_usernames": ["FormerEmployee"],
    "permission_denied_usernames": [],
    "permission_denied_ids": [],
    "unrecognized_parameters": [],
    "permission_denied_usernames": [],
    "unrecognized_parameters": [],
    "other_tokens_revoked": true
  }
}
```

DELETE /tokens/<token>

Use this endpoint to revoke a single token, ensuring that it can no longer be used with RBAC. Authentication is required.

This endpoint is equivalent to `DELETE /tokens?revoke_tokens={TOKEN}`.

This API can be used only by the admin or API user.

Response codes

The server uses the following response codes:

Code	Definition
204 No Content	Sent if all operations are successful.
400 Malformed	Sent if the token is malformed.

The error response is identical to `DELETE /tokens`.

Example error body

The error response from this endpoint is identical to `DELETE /tokens`.

```
{
  "kind": "malformed-request",
  "msg": "The following token is malformed: notAToken. This can be caused by an error while copying and pasting. No tokens were revoked.",
  "details": {
    "malformed_tokens": ["notAToken"],
    "malformed_labels": [],
    "malformed_usernames": [],
    "nonexistent_usernames": [],
    "permission_denied_usernames": [],
    "unrecognized_parameters": [],
    "other_tokens_revoked": false
  }
}
```

POST /auth/token/authenticate

Use this endpoint to exchange a token for a map representing an RBAC subject and associated token data. This endpoint does not require authentication.

This endpoint accepts a JSON body containing entries for `token` and `update_last_activity?`. The `token` field is a string containing an authentication token. The `update_last_activity?` field is a Boolean data type that indicates whether a successful authentication should update the `last_active` timestamp for the token.

Response codes

The server uses the following response codes:

Code	Definition
200 OK	The subject represented by the token.
400 invalid-token	The provided token was either tampered with or could not be parsed.
403 token-revoked	The provided token has been revoked.
403 token-expired	The token has expired and is no longer valid.
403 token-timed-out	The token has timed out due to inactivity.

Example JSON body

```
{
  "token": "0VZZ6geJQK8zJGKxBdqlatTsMLAsfCQFJuRwxsMNgCr4",
  "update_last_activity?": false
}
```

Example return

```
{ "description": null,
  "creation": "YYYY-MM-DDT22:24:30Z",
  "email": "franz@kafka.com",
  "is_revoked": false,
  "last_active": "YYYY-MM-DDT22:24:31Z",
  "last_login": "YYYY-MM-DDT22:24:31.340Z",
  "expiration": "YYYY-MM-DDT22:29:30Z",
  "is_remote": false,
  "client": null,
  "login": "franz@kafka.com",
  "is_superuser": false,
  "label": null,
  "id": "c84bae61-f668-4a18-9a4a-5e33a97b716c",
  "role_ids": [1, 2, 3],
  "user_id": "c84bae61-f668-4a18-9a4a-5e33a97b716c",
  "timeout": null,
  "display_name": "Franz Kafka",
  "is_group": false }
```


Activity service API

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

- [Forming activity service API requests](#) on page 257

Token-based authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or allowed certificates.

- [Event types reported by the activity service](#) on page 258

Activity reporting provides a useful audit trail for actions that change role-based access control (RBAC) entities, such as users, directory groups, and user roles.

- [Events endpoints](#) on page 260

Use the `events` endpoints to retrieve activity service events.

Related information

[API index](#) on page 26

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

Forming activity service API requests

Token-based authentication is required to access the activity service API. You can authenticate requests with user authentication tokens or allowed certificates.

By default, the activity service listens on port 4433. All endpoints are relative to the `/activity-api/` path. So, for example, the full URL for the `/v1/events` endpoint on localhost is `https://localhost:4433/activity-api/v1/events`.

Authentication using tokens

Insert a user authentication token variable in an activity service API request.

1. Generate a token: `puppet-access login`
2. Use `puppet-access show` to populate an authentication header, and use that header in your API request.

```
puppet-access login

auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/activity-api/v1/events?
service_id=classifier"

curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Authenticating using an allowed certificate

You can also authenticate requests using a certificate listed in RBAC's certificate allowlist, located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. Note that if you edit this file, you must reload the `pe-console-services` service (run `sudo service pe-console-services reload`) for your changes to take effect.

Attach the certificate using the command line, as demonstrated in the example curl query below. You must have the allowed certificate name (which must match a name in the `/etc/puppetlabs/console-services/rbac-certificate-allowlist` file) and the private key to run the script.

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
```

```
uri="https://$(puppet config print server):4433/activity-api/v1/events?
service_id=classifier"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
"$uri"
```

You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate specifically for use with the API.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Event types reported by the activity service

Activity reporting provides a useful audit trail for actions that change role-based access control (RBAC) entities, such as users, directory groups, and user roles.

Local users

These events are displayed in the console on the **Activity** tab for the affected user.

Event	Description	Example
Creation	A new local user is created. An initial value for each metadata field is reported.	Created with login set to "jean".
Metadata	Any change to the login, display name, or email keys.	Display name set to "Jean Jackson".
Role membership	A user is added to or removed from a role. The display name and user ID of the affected user are displayed.	User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba) added to role Operators.
Authentication	A user logs in. The display name and user ID of the affected user are displayed.	User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba) logged in.
Password reset token	A token is generated for a user to use when resetting their password. The display name and user ID of the affected user are shown.	A password reset token was generated for user Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba).
Password changed	A user successfully changes their password with a token.	Password reset for user Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba).
Revocation	A user is revoked or reinstated.	User revoked.

Remote users

These events are displayed in the console on the **Activity** tab for the affected user.

Event	Description	Example
Role membership	A user is added to or removed from a role. These events are also shown on the page for the role. The display name and user ID of the affected user are displayed.	User Kalo Hill (76483e62-5ed4-11e4-aa15-123b93f75cba) added to role Viewers.
Revocation	A user is revoked or reinstated.	User revoked.

Directory groups

These events are displayed in the console on the **Activity** tab for the affected group.

Event	Description	Example
Importation	A directory group is imported. The initial value for each metadata field is reported (these cannot be updated using the RBAC UI).	Created with display name set to "Engineers".
Role membership	A group is added to or removed from a role. These events are also shown on the page for the role. The group's display name and ID are provided.	Group Engineers (7dee3acc-5ed4-11e4-aa15-123b93f75cba) added to role Operators.

Roles

These events are displayed in the console on the **Activity** tab for the affected role.

Event	Description	Example
Metadata	A role's display name or description changes.	Description set to "Sysadmins with full privileges for node groups."
Members	A group is added to or removed from a role. The display name and ID of the user or group are provided. These events are also displayed on the page for the affected user or group.	User Kalo Hill (76483e62-5ed4-11e4-aa15-123b93f75cba) removed from role Operators.
Permissions	A permission is added to or removed from a role.	Permission users:edit:76483e62-5ed4-11e4-aa15-123b93f75cba added to role Operators.
Delete	A role has been removed.	The Delete event is recorded and available only through the activity service API, not the Activity tab.

Orchestration

These events are displayed in the console on the **Activity** tab for the affected node.

Event	Description	Example
Agent runs	Puppet runs as part of an orchestration job. This includes runs started from the orchestrator or the PE console.	Request Puppet agent run on node.example.com via orchestrator job 12.
Task runs	Tasks run as orchestration jobs set up in the console or on the command line.	Request echo task on neptune.example.com via orchestrator job 9,607

Authentication tokens

These events are displayed in the console on the **Activity** tab on the affected user's page.

Event	Description	Example
Creation	A new token is generated. These events are exposed in the console on the Activity tab for the user who owns the token.	Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c) generated an authentication token.
Direct revocation	A successful token revocation request. These events are exposed in the console on the Activity tab for the user performing the revocation.	Administrator (42bf351c-f9ec-40af-84ad-e976fec7f4bd) revoked an authentication token belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c), issued at 2016-02-17T21:53:23.000Z and expiring at 2016-02-17T21:58:23.000Z.
Revocation by username	All tokens for a username are revoked. These events are exposed in the console on the Activity tab for the user performing the revocation.	Administrator (42bf351c-f9ec-40af-84ad-e976fec7f4bd) revoked all authentication tokens belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c).

Directory service settings

These events are not exposed in the console. The activity service API must be used to see these events.

Event	Description	Example
Update settings (except password)	A setting is changed in the directory service settings.	User rdn set to "ou=users".
Update directory service password	The directory service password is changed.	Password updated.

Events endpoints

Use the `events` endpoints to retrieve activity service events.

GET /v1/events

Fetches activity service events. Web session authentication is required.

Request format

The /v1/events endpoint supports filtering through query parameters.

Parameter	Value
service_id	The ID of the service. Required.
subject_type	The subject who performed the activity. Required only when subject_id is provided.
subject_id	Comma-separated list of IDs associated with the defined subject type. Optional.
object_type	The object affected by the activity. Required only when the object_id is provided.
object_id	Comma-separated list of IDs associated with the defined object type. Optional.
offset	Number of commits to skip before returning events. Optional.
limit	Maximum number of events to return. Default is 1000 events. Optional.
after_service_commit_time	Return results after a specific service commit time. ISO 8601. Optional.

Response format

Responses are returned in a structured JSON format.

GET /v1/events?service_id=classifier&subject_type=users&subject_id=kai

```
{
  "commits": [
    {
      "object": {
        "id": "415dfsvdf-dfgd45dfg-4dsfg54d",
        "name": "Default Node Group"
      },
      "subject": {
        "id": "dfgdfc145-545dfg54f-fdg45s5s",
        "name": "Kai Evans"
      },
      "timestamp": "2014-06-24T04:00:00Z",
      "events": [
        {
          "message": "Create Node"
        },
        {
          "message": "Create Node Class"
        }
      ]
    }
  ],
  "total-rows": 1
}
```

GET /v1/events?service_id=classifier&object_type=node_groups&object_id=2

```
{
  "commits": [
    {
      "object": {
        "id": "415dfsddf-dfgd45dfg-4dsfg54d",
        "name": "Default Node Group"
      },
      "subject": {
        "id": "dfgdfc145-545dfg54f-fdg45s5s",
        "name": "Kai Evans"
      },
      "timestamp": "2014-06-24T04:00:00Z",
      "events": [
        {
          "message": "Create Node"
        },
        {
          "message": "Create Node Class"
        }
      ]
    }
  ],
  "total-rows": 1
}
```

GET /v1/events.csv

Fetches activity service events and returns in a flat CSV format. Token-based authentication is required.

Request format

The /v1/events.csv endpoint supports accepts the following parameters.

Paramter	Definition
service_id	The ID of the service. Required.
subject_type	The subject who performed the activity. Required only when subject_id is provided.
subject_id	Command-separated list of IDs associated with the defined subject type. Optional.
object_type	The object affected by the activity. Required only when the object_id is provided.
object_id	Comma-separated list of IDs associated with the defined object type. Optional.
offset	Number of commits to skip before returning events. Optional.
limit	Maximum number of events to return. Default is 10000 events. Optional.

Response format

Responses are returned in a flat CSV format. Example return:

```
Submit Time,Subject Type,Subject Id,Subject Name,Object Type,Object
Id,Object Name,Type,What,Description,Message
```

```

YYYY-MM-DD
  18:52:27.76,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,delete,node_group,delete_node_group,"Deleted
the "web_servers" group with id b55c209d-e68f-4096-9a2c-5ae52dd2500c"
YYYY-MM-DD
  18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,create,node_group,create_node_group,"Created
the "web_servers" group with id b55c209d-e68f-4096-9a2c-5ae52dd2500c"
YYYY-MM-DD
  18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_description,edit_node_group_des
the description to """"
YYYY-MM-DD
  18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_environment,edit_node_group_env
the environment to "production"
YYYY-MM-DD
  18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_environment_override,edit_node_g
the environment override setting to false
YYYY-MM-DD
  18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-
e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_parent,edit_node_group_parent,Ch
the parent to ec519937-8681-43d3-8b74-380d65736dba
YYYY-MM-DD
  00:41:18.944,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,ec519937-
Orchestrator,edit,node_group_class_parameter,delete_node_group_class_parameter_puppet_e
the "use_application_services" parameter from the
"puppet_enterprise::profile::orchestrator" class"
YYYY-MM-DD
  00:41:10.631,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,ec519937-
Orchestrator,edit,node_group_class_parameter,add_node_group_class_parameter_puppet_ente
the "use_application_services" parameter to the
"puppet_enterprise::profile::orchestrator" class"
YYYY-MM-DD
  20:41:30.223,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,46e34005-
bc48-4813221e9ffb,PE
Agent,schedule_deploy,node_group,schedule_puppet_agent_on_node_group,Schedule
puppet agent run on nodes in this group to be run at 2019-01-16T08:00:00Z

```

GET /v2/events

Fetches events in a structured JSON format. Allows filtering through query parameters and supports multiple objects for filtering results. Requires token based authentication.

Request format

The `/v2/events` endpoint allows multiple optional objects for filtering results. You can specify as many parameters as you want to filter results. Filtering uses "or" logic, except for the timestamp ranges which use "and" logic.

Parameter	Definition
<code>service_id</code>	The ID of the service. Optional.
<code>offset</code>	Number of commits to skip before returning events. Optional.
<code>limit</code>	Maximum number of events to return. Default is 1000 events. Optional.

Parameter	Definition
query	A JSON-encoded array of optional objects used to filter the results. You can specify multiple objects of the same or different types with the exception of <code>ip_address</code> . You can only specify one <code>ip_address</code> . Results returned based on this logic: (and (or , ,) (or))]. Optional.

The query parameter supports one or more of following:

Parameter	Definition
subject_id	ID associated with the defined subject type. Must be paired with <code>subject_type</code> . Optional.
subject_type	The subject who performed the activity. Defaults to <code>users</code> . Required only when <code>subject_id</code> is provided.
object_id	ID associated with the defined object type. If used, it must be paired with <code>object_type</code> . Optional.
object_type	The object affected by the activity. Optional.
ip_address	Specifies the ip address. Partial string match. Optional.
start	ISO 8601 timestamp for start time. Uses "and" logic and must be paired with <code>end</code> . Optional.
end	ISO 8601 timestamp for end time. Uses "and" logic and must be paired with <code>start</code> . Optional.

For example, this curl request returns all events performed by users 42bf351c-f9ec-40af-84ad-e976fec7f4bd and 95c6159e-dfa3-4f34-b884-45e2ba39d24b from 01 November 2019 through 01 December 2019.

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/activity-api/v2/events?service_id=classifier"
data='query=[{"object_id": "db2cacal-d6a4-4145-8240-9de9b4e654d1", "object_type": "users"},
  {"subject_id": "db2cacal-d6a4-4145-8240-9de9b4e654d1"},
  {"object_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
    "object_type": "users"},
  {"subject_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8"},
  {"start": "2019-11-01T21:32:39Z", "end": "2019-12-01T00:00:00Z"}]'
curl --insecure --header "$auth_header" --request GET "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response format

Responses are returned in a structured JSON format.

For example, the above request produces the following response:

```
{
  "commits": [
    {
      "objects": [
        {
          "id": "5a359d65-a8e5-41f5-b99d-3f0e6b5d668d",
```



```

        "name": "PE Agent",
        "type": "node_groups"
    }
],
"subject": {
    "id": "db2cacal-d6a4-4145-8240-9de9b4e654d1",
    "name": "kai.evans"
},
"timestamp": "2019-11-19T16:40:12Z",
"events": [
    {
        "message": "Added the \"package_inventory_enabled\" parameter to
the \"puppet_enterprise::profile::agent\" class",
        "type": "edit",
        "what": "node_group_class_parameter",
        "description":
"add_node_group_class_parameter_puppet_enterprise::profile::agent_package_inventory_ena
    }
]
},
{
    "objects": [
        {
            "id": "6977ec72-5be3-4e0e-975e-a8d144b9f7ea",
            "name": "test",
            "type": "node_groups"
        }
    ],
    "subject": {
        "id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
        "name": "jean.jackson"
    },
    "timestamp": "2019-11-18T19:43:51Z",
    "events": [
        {
            "message": "Changed the rule to nil",
            "type": "edit",
            "what": "node_group_rule",
            "description": "edit_node_group_rule"
        }
    ]
},
{
    "objects": [
        {
            "id": "6977ec72-5be3-4e0e-975e-a8d144b9f7ea",
            "name": "test",
            "type": "node_groups"
        }
    ],
    "subject": {
        "id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
        "name": "jean.jackson"
    },
    "timestamp": "2019-11-18T19:42:21Z",
    "events": [
        {
            "message": "Added the \"content\" parameter to the \"motd\"
class",
            "type": "edit",
            "what": "node_group_class_parameter",
            "description": "add_node_group_class_parameter_motd_content"
        }
    ]
}

```

```

      "message": "Changed the rule to [\"and\" [\"~\" [\"fact\" \"os\"
\\\"release\" \"major\"] \\\"]]",
      "type": "edit",
      "what": "node_group_rule",
      "description": "edit_node_group_rule"
    }
  ],
},
{
  "objects": [
    {
      "id": "6977ec72-5be3-4e0e-975e-a8d144b9f7ea",
      "name": "test",
      "type": "node_groups"
    }
  ],
  "subject": {
    "id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
    "name": "jean.jackson"
  },
  "timestamp": "2019-11-18T19:41:39Z",
  "events": [
    {
      "message": "Changed the rule to nil",
      "type": "edit",
      "what": "node_group_rule",
      "description": "edit_node_group_rule"
    }
  ]
},
{
  "objects": [
    {
      "id": "6977ec72-5be3-4e0e-975e-a8d144b9f7ea",
      "name": "test",
      "type": "node_groups"
    }
  ],
  "subject": {
    "id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
    "name": "jean.jackson"
  },
  "timestamp": "2019-11-18T19:38:39Z",
  "events": [
    {
      "message": "Added the \"motd\" class",
      "type": "edit",
      "what": "node_group_class",
      "description": "add_node_group_class_motd"
    },
    {
      "message": "Changed the rule to [\"and\" [\"~\" [\"trusted\"
\\\"certname\" \\\"]]",
      "type": "edit",
      "what": "node_group_rule",
      "description": "edit_node_group_rule"
    }
  ]
},
{
  "objects": [
    {
      "id": "5d355a45-cf05-4466-97e4-a1e82a5642b0",
      "name": "Remote Access",

```

```

        "type": "node_groups"
      }
    ],
    "subject": {
      "id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8",
      "name": "jean.jackson"
    },
    "timestamp": "2019-11-01T21:51:03Z",
    "events": [
      {
        "message": "Added the \"profile::client_configs\" class",
        "type": "edit",
        "what": "node_group_class",
        "description": "add_node_group_class_profile::client_configs"
      },
      {
        "message": "Added the \"profile::remote_mgmt\" class",
        "type": "edit",
        "what": "node_group_class",
        "description": "add_node_group_class_profile::remote_mgmt"
      },
      {
        "message": "Added the \"profile::sssd_ldap\" class",
        "type": "edit",
        "what": "node_group_class",
        "description": "add_node_group_class_profile::sssd_ldap"
      },
      {
        "message": "Changed the rule to [\"or\" [\"=\" \"name\" \"pe-
next.p9.puppet.net\"]]",
        "type": "edit",
        "what": "node_group_rule",
        "description": "edit_node_group_rule"
      }
    ]
  }
},
"pagination": {
  "total": 9,
  "limit": 1000,
  "offset": 0
}
}

```

Other examples

Request:

```

GET /v2/events?service_id=classifier&query=[{"subject_id":
"dfgdfcl45-545dfg54f-fdg45s5s", "subject_type": "users"}, {"object_id":
"415dfsvdf-dfgd45dfg-4dsfg54d", "object_type": "node_group"}]

```

Response:

```

{
  "commits": [
    {
      "objects": [{
        "id": "415dfsvdf-dfgd45dfg-4dsfg54d",
        "name": "Default Node Group"
      }],
      "subject": {

```

```

      "id": "dfgdfc145-545dfg54f-fdg45s5s",
      "name": "Kai Evans"
    },
    "timestamp": "2014-06-24T04:00:00Z",
    "events": [
      {
        "message": "Create Node"
      },
      {
        "message": "Create Node Class"
      }
    ]
  }
],
"pagination": {"total": "1", "limit": "1000", "offset": "0"}
}

```

Request:

```
GET /v2/events?service_id=classifier&query=[{"object_id": "415dfsvdf-
dfgd45dfg-4dsfg54d", "object_type": "node_group"}]
```

Response:

```

{
  "commits": [
    {
      "objects": [{
        "id": "415dfsvdf-dfgd45dfg-4dsfg54d",
        "name": "Default Node Group"
      }],
      "subject": {
        "id": "dfgdfc145-545dfg54f-fdg45s5s",
        "name": "Kai Evans"
      },
      "timestamp": "2014-06-24T04:00:00Z",
      "events": [
        {
          "message": "Create Node"
        },
        {
          "message": "Create Node Class"
        }
      ]
    }
  ],
  "pagination": {"total": "1", "limit": "1000", "offset": "0"}
}

```

GET /v2/events.csv

Fetches events in a flat CSV format. Allows filtering through query parameters and supports multiple objects for filtering results. Requires token based authentication.

Request format

The `/v2/events.csv` endpoint allows multiple optional objects for filtering results. You can specify as many parameters as you want to filter results. Filtering uses "or" logic, except for the timestamp ranges which use "and" logic.

Parameter	Definition
service_id	The ID of the service. Optional.
offset	Number of commits to skip before returning events. Optional.
limit	Maximum number of events to return. Default is 1000 events. Optional.
query	A JSON-encoded array of optional objects used to filter the results. Multiple objects of the same or different types can be specified. Results returned based on this logic: (and (or , ,) (or))]. Optional.

The query parameter supports one or more of following:

Parameter	Definition
subject_id	ID associated with the defined subject type. Must be paired with subject_type. Optional.
subject_type	The subject who performed the activity. Defaults to users. Required only when subject_id is provided.
object_id	ID associated with the defined object type. Must be paired with object_type. Optional.
object_type	The object affected by the activity. Required only when the object_id is provided.
ip_address	Specifies the ip address. Optional.
start	ISO 8601 timestamp for start time. Uses "and" logic and must be paired with end. Optional.
end	ISO 8601 timestamp for end time. Uses "and" logic and must be paired with start. Optional.

Response format

GET /v2/events.csv?service_id=classifier&query=[{"subject_id": "kai", "subject_type": "users"}]

```
Submit Time,Subject Type,Subject Id,Subject Name,Object Type,Object
Id,Object Name,Type,What,Description,Message,Ip Address
2014-07-17 13:08:09.985221,users,kai,Kai Evans,node\_groups,2,Default Node
Group,create,node,create\_node,Create Node,123.123.123.123
2014-07-17 13:08:09.985221,users,kai,Kai Evans,node\_groups,2,Default
Node Group,create,node\_class,create\_node\_class,Create Node
Class,123.123.123.123
```

Monitoring and reporting

The console offers a variety of tools you can use to monitor the current state of your infrastructure, see the results of planned or unplanned changes to your Puppet code, view reports, and investigate problems. These tools are grouped in the **Enforcement** section of the console's sidebar.

- [Monitoring infrastructure state](#) on page 270

When nodes fetch their configurations from the primary server, they send back inventory data and a report of their run. This information is summarized on the **Status** page in the console.

- [Viewing and managing packages](#) on page 275

The **Packages** page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the **Packages** page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals.

- [Value report](#) on page 277

Value analytics give you insight into time and money freed by PE automation. You can access these analytics by viewing the **Value report** page in the console or using the value API.

- [Infrastructure reports](#) on page 281

Each time Puppet runs on a node, it generates a report that provides information such as when the run took place, any issues encountered during the run, and the activity of resources on the node. These reports are collected on the **Reports** page in the console.

- [Analyzing changes across Puppet runs](#) on page 285

The **Events** page in the console shows a summary of activity in your infrastructure. You can analyze the details of important changes, and investigate common causes behind related events. You can also examine specific class, node, and resource events, and find out what caused them to fail, change, or run as no-op.

- [Viewing and managing Puppet Server metrics](#) on page 287

Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

- [Status API](#) on page 296

The status API allows you to check the health of PE components and services. It can be useful for automated monitoring of your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

Monitoring infrastructure state

When nodes fetch their configurations from the primary server, they send back inventory data and a report of their run. This information is summarized on the **Status** page in the console.

The **Status** page displays the most recent run status of each of your nodes so you can quickly find issues and diagnose their causes. You can also use this page to gather essential information about your infrastructure at a glance, such as how many nodes your primary server is managing, and whether any nodes are unresponsive.

Node run statuses

The **Status** page displays the run status of each node following the most recent Puppet run. There are 10 possible run statuses.

Nodes run in enforcement mode



With failures

This node's last Puppet run failed, or Puppet encountered an error that prevented it from making changes.

The error is usually tied to a particular resource (such as a file) managed by Puppet on the node. The node as a whole might still be functioning normally. Alternatively, the problem might be caused by a situation on the primary server, preventing the node's agent from verifying whether the node is compliant.



With corrective changes

During the last Puppet run, Puppet found inconsistencies between the last applied catalog and this node's configuration, and corrected those inconsistencies to match the catalog.

Note: Corrective change reporting is available only on agent nodes running PE 2016.4 and later. Agents running earlier versions report all change events as "with intentional changes."



With intentional changes

During the last Puppet run, changes to the catalog were successfully applied to the node.



Unchanged

This node's last Puppet run was successful, and it was fully compliant. No changes were necessary.

Nodes run in no-op mode

Note: No-op mode reporting is available only on agent nodes running PE 2016.4 and later. Agents running earlier versions report all no-op mode runs as "would be unchanged."



With failures

This node's last Puppet run in no-op mode failed, or Puppet encountered an error that prevented it from simulating changes.



Would have corrective changes

During the last Puppet run, Puppet found inconsistencies between the last applied catalog and this node's configuration, and would have corrected those inconsistencies to match the catalog.



Would have intentional changes

During the last Puppet run, catalog changes would have been applied to the node.



Would be unchanged

This node's last Puppet run was successful, and the node was fully compliant. No changes would have been necessary.

Nodes not reporting



Unresponsive

The node hasn't reported to the primary server recently. Something might be wrong. The cutoff for considering a node unresponsive defaults to one hour, and can be configured via the `puppet_enterprise::console_services::no_longer_reporting_cutoff` parameter. See [Configure the PE console and console-services](#) on page 171 for more information.

Check the run status table to see the timestamp for the last known Puppet run for the node and an indication of whether its last known run was in no-op mode. Correct the problem to resume Puppet runs on the node.



Have no reports

Although Puppet Server is aware of this node's existence, the node has never submitted a Puppet report for one or more of the following reasons: it's a newly commissioned node; it has never come online; or its copy of Puppet is not configured correctly.

Note: Expired or deactivated nodes are displayed on the **Status** page for seven days. To extend the amount of time that you can view or search for these nodes, change the `node-ttl` setting in PuppetDB. Changing this setting affects resources and exported resources.

Special categories

In addition to reporting the run status of each node, the **Status** page provides a secondary count of nodes that fall into special categories.

Intended catalog failed

During the last Puppet run, the intended catalog for this node failed, so Puppet substituted a cached catalog, as per your configuration settings.

This typically occurs when you have compilation errors in your Puppet code. Check the Puppet run's log for details.

This category is shown only if one or more agents fail to retrieve a valid catalog from Puppet Server.

Enforced resources found





During the last Puppet run in no-op mode, one or more resources was enforced, as per your use of the `noop => false` metaparameter setting.

This category is shown only if enforced resources are present on one or more nodes.

How Puppet determines node run statuses

Puppet uses a hierarchical system to determine a single run status for each node. This system gives higher priority to the activity types most likely to cause problems in your deployment, so you can focus on the nodes and events most in need of attention.

During a Puppet run, several activity types might occur on a single node. A node's run status reflects the activity with the highest alert level, regardless of how many events of each type took place during the run. Failure events receive the highest alert level, and no change events receive the lowest.

Run status	Definitely happened	Might also have happened
	Failure	Corrective change, intentional change, no change
	Corrective change	Intentional change, no change
	Intentional change	No change
	No change	

For example, during a Puppet run in enforcement mode, a node with 100 resources receives intentional changes on 30 resources, corrective changes on 10 resources, and no changes on the remaining 60 resources. This node's run status is "with corrective changes."

Node run statuses also prioritize run mode (either enforcement or no-op) over the state of individual resources. This means that a node run in no-op mode is always reported in the **Nodes run in no-op** column, even if some of its resource changes were enforced. Suppose the no-op flags on a node's resources are all set to false. Changes to the resources are enforced, not simulated. Even so, because it is run in no-op mode, the node's run status is "would have intentional changes."

Filtering nodes on the Status page

You can filter the list of nodes displayed on the **Status** page by run status and by node fact. If you set a run status filter, and also set a node fact filter, the table takes both filters into account, and shows only those nodes matching both filters.

Clicking **Remove filter** removes all filters currently in effect.

The filters you set are persistent. If you set run status or fact filters on the **Status** page, they continue to be applied to the table until they're changed or removed, even if you navigate to other pages in the console or log out. The persistent storage is associated with the browser tab, not your user account, and is cleared when you close the tab.

Important: The filter results count and the fact filter matching nodes counts are cached for two minutes after first retrieval. This reduces the total load on PuppetDB and decreases page load time, especially for fact filters with multiple rows. As a result, the displayed counts might be up to two minutes out of date.

Filter by node run status

The status counts section at the top of the **Status** page shows a summary of the number of nodes with each run status as of the last Puppet run. Filter nodes by run status to quickly focus on nodes with failures or change events.

In the status counts section, select a run status (such as **with corrective changes** or **have no reports**) or a run status category (such as **Nodes run in no-op**).

Filter by node fact

You can create a highly specific list of nodes for further investigation by using the fact filter tool.

For example, you can check that nodes you've updated have successfully changed, or find out the operating systems or IP addresses of a set of failed nodes to better understand the failure. You might also filter by facts to fulfill an auditor's request for information, such as the number of nodes running a particular version of software.

1. Click **Filter by fact value**. In the **Fact** field, select one of the available facts. An empty fact field is not allowed.

Tip: To see the facts and values reported by a node on its most recent run, click the node name in the **Run status** table, then select the node's **Facts** tab.

2. Select an Operator:

Operator	Meaning	Notes
=	is	
!=	is not	
~	matches a regular expression (regex)	Select this operator to use wildcards and other regular expressions if you want to find matching facts without having to specify the exact value.
!~	does not match a regular expression (regex)	
>	greater than	Can be used only with facts that have a numeric value.
>=	greater than or equal to	Can be used only with facts that have a numeric value.
<	less than	Can be used only with facts that have a numeric value.
<=	less than or equal to	Can be used only with facts that have a numeric value.

3. In the **Value** field, enter a value. Strings are case-sensitive, so make sure you use the correct case.

The filter displays an error if you use an invalid string operator (for example, selecting a numeric value operator such as `>=` and entering a non-numeric string such as `pilsen` as the value) or enter an invalid regular expression.

Note: If you enter an invalid or empty value in the **Value** field, PE takes the following action in order to avoid a filter error:

- Invalid or empty Boolean facts are processed as **false**, and results are retrieved accordingly.
- Invalid or empty numeric facts are processed as **0**, and results are retrieved accordingly.
- Invalid or incomplete regular expressions invalidate the filter, and no results are retrieved.

4. Click **Add**.

5. As needed, repeat these steps to add additional filters. If filtering by more than one node fact, specify either **Nodes must match all rules** or **Nodes can match any rule**.

Filtering nodes in your node list

Filter your node list by node name or by PQL query to more easily inspect them.

Filter your node list by node name

Filter your nodes list by node name to inspect them as a group.

Select **Node name**, type in the word you want to filter by, and click **Submit**.

Filter your nodes by PQL query

Filter your nodes list using a common PQL query.

Filtering your nodes list by PQL query enables you to manage them by specific factors, such as by operating system, report status, or class.

Specify a target by doing one of the following:

- Enter a query that selects the target you want. See the [Puppet Query Language \(PQL\) reference](#) for more information.
- Click **Common queries**. Select one of the queries and replace the defaults in the braces (`{ }`) with values that specify the target you want.

Target	PQL query
All nodes	<code>nodes[certname] { }</code>
Nodes with a specific resource (example: httpd)	<code>resources[certname] { type = "Service" and title = "httpd" }</code>
Nodes with a specific fact and value (example: OS name is CentOS)	<code>inventory[certname] { facts.os.name = "<OS>" }</code>
Nodes with a specific report status (example: last run failed)	<code>reports[certname] { latest_report_status = "failed" }</code>
Nodes with a specific class (example: Apache)	<code>resources[certname] { type = "Class" and title = "Apache" }</code>
Nodes assigned to a specific environment (example: production)	<code>nodes[certname] { catalog_environment = "production" }</code>
Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)	<code>resources[certname] { type = "Package" and title="openssl" and parameters.ensure = "1.0.1e-51.el7_2.7" }</code>

Target	PQL query
Nodes with a specific resource and operating system (example: httpd and CentOS)	<pre>inventory[certname] { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }</pre>

Monitor PE services

PE includes console and command line tools for monitoring the status of core services.

Component or service	Status monitor	Status command
Activity service	#	#
Agentless Catalog Executor (ACE) service		#
Bolt service		#
Classifier service	#	#
Code Manager service	#	#
Orchestrator service	#	#
Puppet Communications Protocol (PCP) broker		#
PostgreSQL		#
Puppet Server	#	#
PuppetDB	#	#
Role-based access control (RBAC) service	#	#

View the Puppet Services status monitor

The **Puppet Services status** monitor provides a visual overview of the current state of core services, and can be used to quickly determine whether an unresponsive or restarting service is causing an issue with your deployment.

A checkmark appears next to **Puppet Services status** if all applicable services are accepting requests. In the event that no data is available, a question mark appears next to the link. If one or more services is restarting or not accepting requests, a warning icon appears.

1. In the console, click **Status**.
2. Click **Puppet Services status** to open the monitor.

puppet infrastructure status command

The `puppet infrastructure status` command displays errors and alerts from PE components and services.

The command reports separately on the primary server and any compilers or replicas in your environment. You must run the command as root.

Viewing and managing packages

The **Packages** page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the **Packages** page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals.

Package inventory reporting is available for all nodes with Puppet agent version 1.6.0 or later installed, including systems that are not actively managed by Puppet.

Tip: Packages are gathered from all available providers. The package data reported on the **Packages** page can also be obtained by using the `puppet resource` command to search for package.

Enable package data collection

Package data collection is disabled by default, so the Packages page in the console initially appears blank. In order to view a node's current package inventory, enable package data collection.

You can choose to collect package data on all your nodes, or just a subset. Any node with Puppet agent version 1.6.0 or later installed can report package data, including nodes that are not under active management with Puppet Enterprise.

1. In the console, click **Node groups**.
 - If you want to collect package data on all your nodes, click the **PE Agent** node group.
 - If you want to collect package data on a subset of your nodes, click **Add group** and create a new classification node group. Select **PE Agent** as the group's parent name. After the new node group is set up, use the **Rules** tab to dynamically add the relevant nodes.
2. Click **Classes**. In the **Add new class** field, select `puppet_enterprise::profile::agent` and click **Add class**.
3. In the `puppet_enterprise::profile::agent` class, set the **Parameter** to `package_inventory_enabled` and the **Value** to `true`. Click **Add parameter**, and commit changes.
4. Run Puppet to apply these changes to the nodes in your node group.

Puppet enables package inventory collection on this Puppet run, and begins collecting package data and reporting it on the **Packages** page on each subsequent Puppet run.
5. Run Puppet a second time to begin collecting package data, then click **Packages**.

View and manage package inventory

To view and manage the complete inventory of packages on your systems, use the Packages page in the console.

Before you begin

Make sure you have enabled package data collection for the nodes you wish to view.

Tip: If all the nodes on which a certain package is installed are deactivated, but the nodes' specified `node-purge-ttl` period has not yet elapsed, instances of the package still appear in summary counts on the **Packages** page. To correct this issue, adjust the `node-purge-ttl` setting and run garbage collection.

1. Run Puppet to collect the latest package data from your nodes.
2. In the console, click **Packages** to view your package inventory. To narrow the list of packages, enter the name or partial name of a package in the **Filter by package name** field and click **Apply**.
3. Click any package name or version to enter the detail page for that package.
4. On a package's detail page, use the **Version** selector to locate nodes with a particular package version installed.
5. Use the **Instances** selector to locate nodes where the package is not managed with Puppet, or to view nodes on which a package instance is managed with Puppet.

To quickly find the place in your manifest where a Puppet-managed package is declared, select a code path in the **Instances** selector and click **Copy path**.

6. To modify a package on a group of nodes:
 - If the package is managed with Puppet, select a code path in the **Instances** selector and click **Copy path**, then navigate to and update the manifest.
 - If the package is not managed with Puppet, click **Run > Task** and create a new task.

View package data collection metadata

The `puppet_inventory_metadata` fact reports whether package data collection is enabled on a node, and shows the time spent collecting package data on the node during the last Puppet run.

Before you begin

Make sure you have enabled package data collection for the nodes you wish to view.

1. Click **Node groups** and select the node group you created when enabling package data collection.
2. Click **Matching nodes** and select a node from the list.
3. On the node's inventory page, click **Facts** and locate **puppet_inventory_metadata** in the list.

The fact value looks something like:

```
{
  "packages" : {
    "collection_enabled" : true,
    "last_collection_time" : "1.9149s"
  }
}
```

Disable package data collection

If you need to disable package data collection, set **package_inventory_enabled** to `false` and run Puppet twice.

1. Click **Node groups** and select the node group you used when enabling package data collection.
2. On the **Classes** tab, find the **puppet_enterprise::profile::agent** class, locate **package_inventory_enabled** parameter, and click **Edit**.
3. Change the **Value** of **package_inventory_enabled** to `false`, then commit changes.
4. Run Puppet to apply these changes to the nodes in your node group and disable package data collection.

Package data is collected for a final time during this run.

5. Run Puppet a second time to purge package data from the impacted nodes' storage.

Value report

Value analytics give you insight into time and money freed by PE automation. You can access these analytics by viewing the **Value report** page in the console or using the value API.

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

Default settings for time freed by each type of automated change are based on customer research. Defaults take into account time to triage, research, and fix issues, as well as context switching. You can specify `low`, `med`, or `high` values for time freed parameters, or provide an exact value based on averages in your business.

Follow these best practices to ensure your value analysis is accurate:

- Failed runs aren't tallied by value analytics, so ensure that tasks, plans, and Puppet runs are processing normally.
- If you are using the API, query the endpoint on a regular basis to gather data over a period of time.

Access the value report

You can use the console to access a snapshot of the time PE automation has reclaimed for you, as well as customize the date and time range PE uses to make these calculations.

Access the **Value report** page in the **Admin** section of the console. The **Value report** page displays the total time reclaimed per node, per day, and over the selected date range.

Click **Configure report** to change the date range or change the average amount of time reclaimed with each action in your environment: intentional changes, correctional changes, task runs, or plan runs. If you're unsure how much time each action reclaims, click **View recommended ranges** for guidance.

GET /api/reports/value

Use the /value endpoint to retrieve information about time and money freed by PE automation.

Forming requests

Requests to the value API must be well-formed HTTP(S) requests.

By default, the value API uses the standard HTTPS port for console communication, port 443. You can omit the port from requests unless you want to specify a different port.

Authenticating

You must authenticate requests to the value API using your Puppet CA certificate and an RBAC token. The RBAC token must have viewing permissions for the console.

A simple, authenticated API request without any parameters looks like this:

```
curl https://<HOSTNAME>/api/reports/value \
  --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
  -H "X-Authentication: <RBAC_TOKEN>"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Query parameters

The request accepts the following parameters:

Parameter	Value	Default
averageHourlySalary	Numeric value specifying average hourly cost savings for automated work.	none
startDate	Date in yyyy-mm-dd format.	1 week ago +2 days
endDate	Date in yyyy-mm-dd format. Specifying the current date results in provisional data.	today -2 days
minutesFreedPerCorrectiveChange	<ul style="list-style-type: none"> low (90 minutes) med (180 minutes) high (360 minutes) any numeric value in minutes 	med

Parameter	Value	Default
minutesFreedPerIntentionalChange	<ul style="list-style-type: none"> low (30 min) med (90 minutes) high (180 minutes) any numeric value in minutes 	med
minutesFreedPerTaskRun	<ul style="list-style-type: none"> low (30 minutes) med (90 minutes) high (180 minutes) any numeric value in minutes 	med
minutesFreedPerPlanRun	<ul style="list-style-type: none"> low (90 minutes) med (180 minutes) high (360 minutes) any numeric value in minutes 	med

Response format

The response is a JSON object that lists details about time and cost freed, using the following keys:

Key	Definition
startDate	Start date for the reporting period.
endDate	End date for the reporting period.
totalCorrectiveChanges	Total number of corrective changes made during the reporting period.
minutesFreedByCorrectiveChanges	Total number of minutes freed by automated changes that prevent drift during regular Puppet runs. The calculation is based on the average minutes saved per change, as specified by the minutesFreedPerCorrectiveChange query parameter.
totalIntentionalChanges	Total number of intentional changes made during the reporting period.
minutesFreedByIntentionalChanges	Total number of minutes freed by automated changes based on new values or Puppet code. This calculation is based on the average minutes saved per change, as specified by the minutesFreedPerIntentionalChange query parameter.
totalNodesAffectedByTaskRuns	Total number of nodes affected by successful task runs during the reporting period.

Key	Definition
minutesFreedByTaskRuns	Total number of minutes freed by automated task runs. This calculation is based on the average minutes saved per task run, as specified by the minutesFreedPerTaskRun query parameter.
totalNodesAffectedByPlanRuns	Total number of nodes affected by successful plan runs during the reporting period.
minutesFreedByPlanRuns	Total number of minutes freed by automated plan runs. This calculation is based on the average minutes saved per plan run, as specified by the minutesFreedPerPlanRun query parameter.
totalMinutesFreed	Total number of minutes free by all automated changes.
totalDollarsSaved	If the query specified an averageHourlySalary, total cost savings for all automated changes.

Examples

To generate a report for specified dates using the default time freed values:

```
curl -G https://<HOSTNAME>/api/reports/value \
--cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
-H "X-Authentication: <RBAC_TOKEN>" \
--data-urlencode 'startDate=2020-07-08' \
--data-urlencode 'endDate=2020-07-15'
```

Result:

```
{
  "startDate": "2020-07-08",
  "endDate": "2020-07-15",
  "totalCorrectiveChanges": 0,
  "minutesFreedByCorrectiveChanges": 0,
  "totalIntentionalChanges": 18,
  "minutesFreedByIntentionalChanges": 1620,
  "totalNodesAffectedByPlanRuns": 0,
  "totalNodesAffectedByTaskRuns": 0,
  "minutesFreedByPlanRuns": 0,
  "minutesFreedByTaskRuns": 0,
  "totalMinutesFreed": 1620
}
```

To generate cost savings using default report dates and time freed values:

```
curl -G https://<pe-console-fqdn>/api/reports/value \
--cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
-H "X-Authentication: <rbac token>" \
--data-urlencode 'averageHourlySalary=40'
```

Result:

```
{
  "startDate": "2020-07-08",
  "endDate": "2020-07-15",
  "totalCorrectiveChanges": 0,
  "minutesFreedByCorrectiveChanges": 0,
  "totalIntentionalChanges": 18,
  "minutesFreedByIntentionalChanges": 1620,
}
```



```

    "totalNodesAffectedByPlanRuns": 0,
    "totalNodesAffectedByTaskRuns": 0,
    "minutesFreedByPlanRuns": 0,
    "minutesFreedByTaskRuns": 0,
    "totalMinutesFreed": 1620,
    "totalDollarsSaved": 1080,
  }

```

To generate a report with custom values for time freed:

```

curl -G https://<pe-console-fqdn>/api/reports/value \
  --cacert /etc/puppetlabs/puppet/ssl/certs/ca.pem \
  -H "X-Authentication: $(cat ~/.puppetlabs/token)" \
  --data-urlencode 'minutesFreedPerCorrectiveChange=10' \
  --data-urlencode 'minutesFreedPerIntentionalChange=20' \
  --data-urlencode 'minutesFreedPerTaskRun=30' \
  --data-urlencode 'minutesFreedPerPlanRun=40'

```

Result:

```

{
  "startDate": "2020-07-01",
  "endDate": "2020-07-08",
  "totalCorrectiveChanges": 1,
  "minutesFreedByCorrectiveChanges": 10,
  "totalIntentionalChanges": 2,
  "minutesFreedByIntentionalChanges": 40,
  "totalNodesAffectedByTaskRuns": 3,
  "minutesFreedByTaskRuns": 90,
  "totalNodesAffectedByPlanRuns": 4,
  "minutesFreedByPlanRuns": 160,
  "totalMinutesFreed": 300
}

```

Infrastructure reports

Each time Puppet runs on a node, it generates a report that provides information such as when the run took place, any issues encountered during the run, and the activity of resources on the node. These reports are collected on the **Reports** page in the console.

Working with the reports table

The **Reports** page provides a summary view of key data from each report. Use this page to track recent node activity so you can audit your system and perform root cause analysis over time.

The reports table lists the number of resources on each node in each of the following states:

Correction applied	Number of resources that received a corrective change after Puppet identified resources that were out of sync with the applied catalog.
Failed	Number of resources that failed.
Changed	Number of resources that changed.
Unchanged	Number of resources that remained unchanged.
No-op	Number of resources that would have been changed if not run in no-op mode.

Skipped	Number of resources that were skipped because they depended on resources that failed.
Failed restarts	<p>Number of resources that were supposed to restart but didn't.</p> <p>For example, if changes to one resource notify another resource to restart, and that resource doesn't restart, a failed restart is reported. It's an indirect failure that occurred in a resource that was otherwise unchanged.</p>

The reports table also offers the following information:

- **No-op mode:** An indicator of whether the node was run in no-op mode.
- **Config retrieval:** Time spent retrieving the catalog for the node (in seconds).
- **Run time:** Time spent applying the catalog on the node (in seconds).

Tip: Report count caching is used to improve console performance. In some cases, caching might cause summary counts of available reports to be displayed inaccurately the first time the page is accessed after a fresh install.

Filtering reports

You can filter the list of reports displayed on the **Reports** page by run status and by node fact. If you set a run status filter, and also set a node fact filter, the table takes both filters into account, and shows only those reports matching both filters.

Clicking **Remove filter** removes all filters currently in effect.

The filters you set are persistent. If you set run status or fact filters on the **Reports** page, they continue to be applied to the table until they're changed or removed, even if you navigate to other pages in the console or log out. The persistent storage is associated with the browser tab, not your user account, and is cleared when you close the tab.

Filter by node run status

Filter reports to quickly focus on nodes with failures or change events by using the **Filter by run status** bar.

1. Select a run status (such as **No-op mode: with failures**). The table updates to reflect your filter selection.
2. To remove the run status filter, select **All run statuses**.

Filter by node fact

You can create a highly specific list of nodes for further investigation by using the fact filter tool.

For example, you can check that nodes you've updated have successfully changed, or find out the operating systems or IP addresses of a set of failed nodes to better understand the failure. You might also filter by facts to fulfill an auditor's request for information, such as the number of nodes running a particular version of software.

1. Click **Filter by fact value**. In the **Fact** field, select one of the available facts. An empty fact field is not allowed.

Tip: To see the facts and values reported by a node on its most recent run, click the node name in the **Run status** table, then select the node's **Facts** tab.

2. Select an Operator:

Operator	Meaning	Notes
=	is	
!=	is not	
~	matches a regular expression (regex)	Select this operator to use wildcards and other regular expressions if you want to find matching facts without having to specify the exact value.
!~	does not match a regular expression (regex)	
>	greater than	Can be used only with facts that have a numeric value.
>=	greater than or equal to	Can be used only with facts that have a numeric value.
<	less than	Can be used only with facts that have a numeric value.
<=	less than or equal to	Can be used only with facts that have a numeric value.

3. In the **Value** field, enter a value. Strings are case-sensitive, so make sure you use the correct case.

The filter displays an error if you use an invalid string operator (for example, selecting a numeric value operator such as `>=` and entering a non-numeric string such as `pilsen` as the value) or enter an invalid regular expression.

Note: If you enter an invalid or empty value in the **Value** field, PE takes the following action in order to avoid a filter error:

- Invalid or empty Boolean facts are processed as **false**, and results are retrieved accordingly.
- Invalid or empty numeric facts are processed as **0**, and results are retrieved accordingly.
- Invalid or incomplete regular expressions invalidate the filter, and no results are retrieved.

4. Click **Add**.

5. As needed, repeat these steps to add additional filters. If filtering by more than one node fact, specify either **Nodes must match all rules** or **Nodes can match any rule**.

Working with individual reports

To examine a report in greater detail, click **Report time**. This opens a page that provides details for the node's resources in three sections: **Events**, **Log**, and **Metrics**.

Events

The **Events** tab lists the events for each managed resource on the node, its status, whether correction was applied to the resource, and — if it changed — what it changed from and what it changed to. For example, a user or a file might change from absent to present.

To filter resources by event type, click **Filter by event status** and choose an event.

Sort resources by name or events by severity level, ascending or descending, by clicking the **Resource** or **Events** sorting controls.

To download the events data as a `.csv` file, click **Export data**. The filename is `events-<node name>-<timestamp>`.

Log

The **Log** tab lists errors, warnings, and notifications from the node's latest Puppet run.

Each message is assigned one of the following severity levels:

Standard	Caution (yellow)	Warning (red)
debug	warning	err
info	alert	emerg
notice		crit

To read the report chronologically, click the time sorting controls. To read it in order of issue severity, click the severity level sorting controls.

To download the log data as a .csv file, click **Export data**. The filename is `log-<node name>-<timestamp>`.

Metrics

The **Metrics** tab provides a summary of the key data from the node's latest Puppet run.

Metric	Description
Report submitted by:	The certname of the primary server that submitted the report to PuppetDB.
Puppet environment	The environment assigned to the node.
Puppet run	<ul style="list-style-type: none"> The time that the Puppet run began The time that the primary server submitted the catalog The time that the Puppet run finished The time PuppetDB received the report The duration of the Puppet run The length of time to retrieve the catalog The length of time to apply the resources to the catalog
Catalog application	Information about the catalog application that produces the report: the config version that Puppet uses to match a specific catalog for a node to a specific Puppet run, the catalog UUID that identifies the catalog used to generate a report during a Puppet run, and whether the Puppet run used a cached catalog.
Resources	The total number of resources in the catalog.
Events	A list of event types and the total count for each one.
Top resource types	A list of the top resource types by time, in seconds, it took to be applied.

Analyzing changes across Puppet runs

The **Events** page in the console shows a summary of activity in your infrastructure. You can analyze the details of important changes, and investigate common causes behind related events. You can also examine specific class, node, and resource events, and find out what caused them to fail, change, or run as no-op.

What is an event?

An event occurs whenever PE attempts to modify an individual property of a given resource. Reviewing events lets you see detailed information about what has changed on your system, or what isn't working properly.

During a Puppet run, Puppet compares the current state of each property on each resource to the desired state for that property, as defined by the node's catalog. If Puppet successfully compares the states and the property is already in sync (in other words, if the current state is the desired state), Puppet moves on to the next resource without noting anything. Otherwise, it attempts some action and records an event, which appears in the report it sends to the primary server at the end of the run. These reports provide the data presented on the **Events** page in the console.

Event types

There are six types of event that can occur when Puppet reviews each property in your system and attempts to make any needed changes. If a property is already in sync with its catalog, no event is recorded: no news is good news in the world of events.

Event	Description
Failure	A property was out of sync; Puppet tried to make changes, but was unsuccessful.
Corrective change	Puppet found an inconsistency between the last applied catalog and a property's configuration, and corrected the property to match the catalog.
Intentional change	Puppet applied catalog changes to a property.
Corrective no-op	Puppet found an inconsistency between the last applied catalog and a property's configuration, but Puppet was instructed to not make changes on this resource, via either the <code>--noop</code> command-line option, the <code>noop</code> setting, or the <code>noop => true</code> metaparameter. Instead of making a corrective change, Puppet logs a corrective no-op event and reports the change it would have made.
Intentional no-op	Puppet would have applied catalog changes to a property., but Puppet was instructed to not make changes on this resource, via either the <code>--noop</code> command-line option, the <code>noop</code> setting, or the <code>noop => true</code> metaparameter. Instead of making an intentional change, Puppet logs an intentional no-op event and reports the change it would have made.

Event	Description
Skip	<p>A prerequisite for this resource was not met, so Puppet did not compare its current state to the desired state. This prerequisite is either one of the resource's dependencies or a timing limitation set with the <code>schedule</code> metaparameter. The resource might be in sync or out of sync; Puppet doesn't know yet..</p> <p>If the <code>schedule</code> metaparameter is set for a given resource, and the scheduled time hasn't arrived when the run happens, that resource logs a skip event on the Events page. This is true for a user-defined <code>schedule</code>, but does not apply to built-in scheduled tasks that happen weekly, daily, or at other intervals.</p>

Working with the Events page

During times when your deployment is in a state of stability, with no changes being made and everything functioning optimally, the **Events** page reports little activity, and might not seem terribly interesting. But when change occurs—when packages require upgrades, when security concerns threaten, or when systems fail—the **Events** page helps you understand what's happening and where so you can react quickly.

The **Events** page fetches data when loading, and does not refresh—even if there's a Puppet run while you're on the page—until you close or reload the page. This ensures that shifting data won't disrupt an investigation.

You can see how recent the shown data is by checking the timestamp at the top of the page. Reload the page to update the data to the most recent events.

Tip: Keeping time synchronized by running NTP across your deployment helps the **Events** page produce accurate information. NTP is easily managed with PE, and setting it up is an excellent way to learn Puppet workflows.

Monitoring infrastructure with the Events summary pane

The **Events** page displays all events from the latest report of every responsive node in the deployment.

Tip: By default, PE considers a node unresponsive after one hour, but you can configure this setting to meet your needs by adjusting the `puppet_enterprise::console_services::no_longer_reporting_cutoff` parameter.

On the left side of the screen, the **Events** summary pane shows an overview of Puppet activity across your infrastructure. This data can help you rapidly assess the magnitude of any issue.

The **Events** summary pane is split into three categories—the **Classes** summary, **Nodes** summary, and **Resources** summary—to help you investigate how a change or failure event impacts your entire deployment.

Gaining insight with the Events detail pane

Clicking an item in the **Events** summary pane loads its details (and any sub-items) in the **Events** detail pane on the right of the screen. The summary pane on the left always shows the list of items from which the one in the detail pane on the right was chosen, to let you easily view similar items and compare their states.

Click any item in the the **Classes** summary, **Nodes** summary, or **Resources** summary to load more specific info into the detail pane and begin looking for the causes of notable events. Switch between perspectives to find the common threads among a group of failures or corrective changes, and follow them to a root cause.

Analyzing changes and failures

You can use the **Events** page to analyze the root causes of events resulting from a Puppet run. For example, to understand the cause of a failure after a Puppet run, select the class, node, or resource with a failure in the **Events** summary pane, and then review the details of the failure in the **Events** detail pane.

You can view additional details by clicking on the failed item in the in the **Events** detail pane.

Use the **Classes** summary, **Nodes** summary, and **Resources** summary to focus on the information you need. For example, if you're concerned about a failed service, say Apache or MongoDB, you can start by looking into failed resources or classes. If you're experiencing a geographic outage, you might start by drilling into failed node events.

Understanding event display issues

In some special cases, events are not displayed as expected on the **Events** page. These cases are often caused by the way that the console receives data from other parts of Puppet Enterprise, but sometimes are due to the way your Puppet code is interpreted.

Runs that restart PuppetDB are not displayed

If a given Puppet run restarts PuppetDB, Puppet is not able to submit a run report from that run to PuppetDB because PuppetDB is not available. Because the **Events** page relies on data from PuppetDB, and PuppetDB reports are not queued, the **Events** page does not display any events from that run. Note that in such cases, a run report *is* available on the **Reports** page. Having a Puppet run restart PuppetDB is an unlikely scenario, but one that could arise in cases where some change to, say, a parameter in the `puppetdb` class causes the `pe-puppetdb` service to restart.

Runs without a compiled catalog are not displayed

If a run encounters a catastrophic failure where an error prevents a catalog from compiling, the **Events** page does not display any failures. This is because no events occurred.

Simplified display for some resource types

For resource types that take the `ensure` property, such as user or file resource types, the **Events** page displays a single event when the resource is first created. This is because Puppet has changed only one property (`ensure`), which sets all the baseline properties of that resource at the same time. For example, all of the properties of a given user are created when the user is added, just as if the user was added manually. If a later Puppet run changes properties of that user resource, each individual property change is shown as a separate event.

Updated modes display without leading zeros

When the `mode` attribute for a `file` resource is updated, and numeric notation is used, leading zeros are omitted in the **New Value** field on the **Events** page. For example, 0660 is shown as 660 and 0000 is shown as 0.

Viewing and managing Puppet Server metrics

Puppet Server can provide performance and status metrics to external services for monitoring server health and performance over time.

You can track Puppet Server metrics by using:

- Customizable, networked Graphite and Grafana instances
- A built-in experimental developer dashboard
- Status API endpoints

Note: None of these methods are officially supported. The Grafanadash and Puppet-graphite modules referenced in this document are not Puppet-supported modules; they are mentioned for testing and demonstration purposes only. The developer dashboard is a tech preview. Both the Grafana and developer dashboard methods take advantage of the Status API, including some endpoints that are also a tech preview.

- [Getting started with Graphite](#) on page 288

Puppet Enterprise can export many metrics to Graphite, a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. After Graphite support is enabled, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

- [Available Graphite metrics](#) on page 292

These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

Getting started with Graphite

Puppet Enterprise can export many metrics to Graphite, a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. After Graphite support is enabled, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

Note: A Graphite setup is deeply customizable and can report many different Puppet Server metrics on demand. However, it requires considerable configuration and additional server resources. For an easier, but more limited, web-based dashboard of Puppet Server metrics built into Puppet Server, use the developer dashboard. To retrieve metrics manually via HTTP, use the Status API.

To use Graphite with Puppet Enterprise, you must:

- Install and configure a Graphite server.
- Enable Puppet Server's Graphite support

Grafana provides a web-based customizable dashboard that's compatible with Graphite, and the Grafanadash module installs and configures it by default.

Using the Grafanadash module

The Grafanadash module quickly installs and configures a basic test instance of Graphite with the Grafana extension. When installed on a dedicated Puppet agent, this module provides a quick demonstration of how Graphite and Grafana can consume and display Puppet Server metrics.



CAUTION: The Grafanadash module referenced in this document is not a Puppet-supported module; it is for testing and demonstration purposes only. It is tested against CentOS 7 only. Also, install this module on a dedicated agent only. Do not install it on the primary server, because the module makes security policy changes that are inappropriate for a primary server:

- SELinux can cause issues with Graphite and Grafana, so the module temporarily disables SELinux. If you reboot the machine after using the module to install Graphite, you must disable SELinux again and restart the Apache service to use Graphite and Grafana.
- The module disables the iptables firewall and enables cross-origin resource sharing on Apache, which are potential security risks.

Installing the Grafanadash module

Install the Grafanadash module on a *nix agent. The module's `grafanadash::dev` class installs and configures a Graphite server, the Grafana extension, and a default dashboard.

1. Install a *nix PE agent to serve as the Graphite server.
2. As root on the Puppet agent node, run `puppet module install puppetlabs-grafanadash`.
3. As root on the Puppet agent node, run `puppet apply -e 'include grafanadash::dev'`.

Running Grafana

Grafana is a dashboard that can interpret and visualize Puppet Server metrics over time, but you must configure it to do so.

Grafana runs as a web dashboard, and the Grafanadash module configures it at port 10000 by default. However, there are no Puppet metrics displayed by default. You must create a metrics dashboard to view Puppet's metrics in Grafana, or edit and import a JSON-based dashboard such as the sample Grafana dashboard that we provide.

1. In a web browser on a computer that can reach the Puppet agent node, navigate to `http://<AGENT_HOSTNAME>:10000`.

2. Open the `sample_metrics_dashboard.json` file in a text editor on the same computer you're using to access Grafana.
3. Throughout the file, replace our sample setting of `primary.example.com` with the hostname of your primary server. This value must be used as the `metrics_server_id` setting, as configured below.
4. Save the file.
5. In the Grafana UI, click **search** (the folder icon), then **Import**, then **Browse**.
6. Navigate to and select the edited JSON file.

This loads a dashboard with nine graphs that display various metrics exported from the Puppet Server to the Graphite server. However, these graphs remain empty until you enable Puppet Server's Graphite metrics.

Related information

[Sample Grafana dashboard graphs](#) on page 289

Use the sample Grafana dashboard as your starting point and customize it to suit your needs. You can click on the title of any graph, and then click **edit** to adjust the graphs as you see fit.

Enabling Puppet Server's Graphite support

Use the **PE Master** node group in the console to configure Puppet Server's metrics output settings.

1. In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
2. On the **Classes** tab, in the `puppet_enterprise::profile::master` class, add these parameters:
 - a) Set `metrics_graphite_enabled` to `true` (default is `false`).
 - b) Set `metrics_server_id` to the primary server hostname.
 - c) Set `metrics_graphite_host` to the hostname for the agent node on which you're running Graphite and Grafana.
 - d) Set `metrics_graphite_update_interval_seconds` to a value to set Graphite's update frequency in seconds. This setting is optional, and the default value is 60.
3. Verify that these parameters are set to their default values, unless your Graphite server uses a non-standard port:
 - a) Set `metrics_jmx_enabled` to `true` (default value).
 - b) Set `metrics_graphite_port` to 2003 (default value) or the Graphite port on your Graphite server.
 - c) Set `profiler_enabled` to `true` (default value).
4. Commit changes.

Sample Grafana dashboard graphs

Use the sample Grafana dashboard as your starting point and customize it to suit your needs. You can click on the title of any graph, and then click **edit** to adjust the graphs as you see fit.

[Sample Grafana dashboard code](#)

Graph name	Description
Active requests	This graph serves as a "health check" for the Puppet Server. It shows a flat line that represents the number of CPUs you have in your system, a metric that indicates the total number of HTTP requests actively being processed by the server at any moment in time, and a rolling average of the number of active requests. If the number of requests being processed exceeds the number of CPUs for any significant length of time, your server might be receiving more requests than it can efficiently process.

Graph name	Description
Request durations	This graph breaks down the average response times for different types of requests made by Puppet agents. This indicates how expensive catalog and report requests are compared to the other types of requests. It also provides a way to see changes in catalog compilation times when you modify your Puppet code. A sharp curve upward for all of the types of requests indicates an overloaded server, and they should trend downward after reducing the load on the server.
Request ratios	This graph shows how many requests of each type that Puppet Server has handled. Under normal circumstances, you should see about the same number of catalog, node, or report requests, because these all happen one time per agent run. The number of file and file metadata requests correlate to how many remote file resources are in the agents' catalogs.
External HTTP Communications	This graph tracks the amount of time it takes Puppet Server to send data and requests for common operations to, and receive responses from, external HTTP services, such as PuppetDB.
File Sync	This graph tracks how long Puppet Server spends on File Sync operations, for both its storage and client services.
JRubies	This graph tracks how many JRubies are in use, how many are free, the mean number of free JRubies, and the mean number of requested JRubies. If the number of free JRubies is often less than one, or the mean number of free JRubies is less than one, Puppet Server is requesting and consuming more JRubies than are available. This overload reduces Puppet Server's performance. While this might simply be a symptom of an under-resourced server, it can also be caused by poorly optimized Puppet code or bottlenecks in the server's communications with PuppetDB if it is in use. If catalog compilation times have increased but PuppetDB performance remains the same, examine your Puppet code for potentially unoptimized code. If PuppetDB communication times have increased, tune PuppetDB for better performance or allocate more resources to it. If neither catalog compilation nor PuppetDB communication times are degraded, the Puppet Server process might be under-resourced on your server. If you have available CPU time and memory, increase the number of JRuby instances to allow it to allocate more JRubies. Otherwise, consider adding additional compilers to distribute the catalog compilation load.

Graph name	Description
JRuby Timers	<p>This graph tracks several JRuby pool metrics.</p> <ul style="list-style-type: none"> • The borrow time represents the mean amount of time that Puppet Server uses ("borrows") each JRuby from the pool. • The wait time represents the total amount of time that Puppet Server waits for a free JRuby instance. • The lock held time represents the amount of time that Puppet Server holds a lock on the pool, during which JRubies cannot be borrowed. This occurs while Puppet Server synchronizes code for File Sync. • The lock wait time represents the amount of time that Puppet Server waits to acquire a lock on the pool. <p>These metrics help identify sources of potential JRuby allocation bottlenecks.</p>
Memory Usage	This graph tracks how much heap and non-heap memory that Puppet Server uses.
Compilation	This graph breaks catalog compilation down into various phases to show how expensive each phase is on the primary server.

Example Grafana dashboard excerpt

The following example shows only the `targets` parameter of a dashboard to demonstrate the full names of Puppet's exported Graphite metrics (assuming the Puppet Server instance has a domain of `primary.example.com`) and a way to add targets directly to an exported Grafana dashboard's JSON content.

```
"panels": [
  {
    "span": 4,
    "editable": true,
    "type": "graphite",
    ...
    "targets": [
      {
        "target": "alias(puppetlabs.primary.example.com.num-
cpus,'num cpus')",
      },
      {
        "target": "alias(puppetlabs.primary.example.com.http.active-
requests.count,'active requests')",
      },
    ],
  },
]
```

```

        {
          "target": "alias(puppetlabs.primary.example.com.http.active-
histo.mean,'average')"
        },
        "aliasColors": {},
        "aliasYAxis": {},
        "title": "Active Requests"
      }
    ]
  ]
}

```

See the sample Grafana dashboard for a detailed example of how a Grafana dashboard accesses these exported Graphite metrics.

Available Graphite metrics

These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

Graphite metrics properties

Each metric is prefixed with `puppetlabs.<PRIMARY_HOSTNAME>`; for instance, the Grafana dashboard file refers to the `num-cpus` metric as `puppetlabs.<PRIMARY_HOSTNAME>.num-cpus`.

Additionally, metrics might be suffixed by fields, such as `count` or `mean`, that return more specific data points. For instance, the `puppetlabs.<PRIMARY_HOSTNAME>.compiler.mean` metric returns only the mean length of time it takes Puppet Server to compile a catalog.

To aid with reference, metrics in the list below are segmented into three groups:

- **Statistical metrics:** Metrics that have all eight of these statistical analysis fields, in addition to the top-level metric:
 - `max`: Its maximum measured value.
 - `min`: Its minimum measured value.
 - `mean`: Its mean, or average, value.
 - `stddev`: Its standard deviation from the mean.
 - `count`: An incremental counter.
 - `p50`: The value of its 50th percentile, or median.
 - `p75`: The value of its 75th percentile.
 - `p95`: The value of its 95th percentile.
- **Counters only:** Metrics that only count a value, or only have a `count` field.
- **Other:** Metrics that have unique sets of available fields.

Note:

Puppet Server can export many metrics—so many that past versions of Puppet Enterprise could overwhelm Grafana servers. As of Puppet Enterprise 2016.4, Puppet Server exports only a subset of its available metrics by default. This set is designed to report the most relevant Puppet Server metrics for administrators monitoring its performance and stability.

To add to the default list of exported metrics, see [Modifying Puppet Server's exported metrics](#).

Puppet Server exports each metric in the lists below by default.

Statistical metrics

Compiler metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.compiler`: The time spent compiling catalogs. This metric represents the sum of the `compiler.compile`, `static_compile`, `find_facts`, and `find_node` fields.
 - `puppetlabs.<PRIMARY_HOSTNAME>.compiler.compile`: The total time spent compiling dynamic (non-static) catalogs.

To measure specific nodes and environments, see [Modifying Puppet Server's exported metrics](#).

- `puppetlabs.<PRIMARY_HOSTNAME>.compiler.find_facts`: The time spent parsing facts.
- `puppetlabs.<PRIMARY_HOSTNAME>.compiler.find_node`: The time spent retrieving node data. If the Node Classifier (or another ENC) is configured, this includes the time spent communicating with it.
- `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile`: The time spent compiling static catalogs.
- `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile_inlining`: The time spent inlining metadata for static catalogs.
- `puppetlabs.<PRIMARY_HOSTNAME>.compiler.static_compile_postprocessing`: The time spent post-processing static catalogs.

File sync metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.clone-timer`: The time spent by file sync clients on compilers initially cloning repositories on the primary server.
- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.fetch-timer`: The time spent by file sync clients on compilers fetching repository updates from the primary server.
- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.sync-clean-check-timer`: The time spent by file sync clients on compilers checking whether the repositories are clean.
- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-client.sync-timer`: The time spent by file sync clients on compilers synchronizing code from the private datadir to the live codedir.
- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-storage.commit-add-rm-timer`
- `puppetlabs.<PRIMARY_HOSTNAME>.file-sync-storage.commit-timer`: The time spent committing code on the primary server into the file sync repository.

Function metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.functions`: The amount of time during catalog compilation spent in function calls. The `functions` metric can also report any of the statistical metrics fields for a single function by specifying the function name as a field.

For example, to report the mean time spent in a function call during catalog compilation, use `puppetlabs.<PRIMARY_HOSTNAME>.functions.<FUNCTION-NAME>.mean`.

HTTP metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.http.active-histo`: A histogram of active HTTP requests over time.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-catalog-/*/-requests`: The time Puppet Server has spent handling catalog requests, including time spent waiting for an available JRuby instance.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment-/*/-requests`: The time Puppet Server has spent handling environment requests, including time spent waiting for an available JRuby instance.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment_classes-/*/-requests`: The time spent handling requests to the `environment_classes` API endpoint, which the Node Classifier uses to refresh classes.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environments-requests`: The time spent handling requests to the `environments` API endpoint requests made by the Orchestrator

- The following metrics measure the time spent handling file-related API endpoints:
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-requests`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_content-/*/-requests`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadata-/*/-requests`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadatas-/*/-requests`
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-node-/*/-requests`: The time spent handling node requests, which are sent to the Node Classifier. A bottleneck here might indicate an issue with the Node Classifier or PuppetDB.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-report-/*/-requests`: The time spent handling report requests. A bottleneck here might indicate an issue with PuppetDB.
- `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-static_file_content-/*/-requests`: The time spent handling requests to the `static_file_content` API endpoint used by Direct Puppet with file sync.

JRuby metrics: Puppet Server uses an embedded JRuby interpreter to execute Ruby code. JRuby spawns parallel instances known as JRubies to execute Ruby code, which occurs during most Puppet Server activities. See [Tuning JRuby on Puppet Server](#) for details on adjusting JRuby settings.

- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-timer`: The time spent with a borrowed JRuby.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.free-jrubies-histo`: A histogram of free JRubies over time. This metric's average value must be greater than 1; if it isn't, more JRubies or another compiler might be needed to keep up with requests.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.lock-held-timer`: The time spent holding the JRuby lock.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.lock-wait-timer`: The time spent waiting to acquire the JRuby lock.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.requested-jrubies-histo`: A histogram of requested JRubies over time. This increases as the number of free JRubies, or the `free-jrubies-histo` metric, decreases, which can suggest that the server's capacity is being depleted.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.wait-timer`: The time spent waiting to borrow a JRuby.

PuppetDB metrics: The following metrics measure the time that Puppet Server spends sending or receiving data from PuppetDB.

- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.catalog.save`
- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.command.submit`
- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.facts.find`
- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.facts.search`
- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.report.process`
- `puppetlabs.<PRIMARY_HOSTNAME>.puppetdb.resource.search`

Counters only

HTTP metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.http.active-requests`: The number of active HTTP requests.

- The following counter metrics report the percentage of each HTTP API endpoint's share of total handled HTTP requests.
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-catalog-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environment_classes-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-environments-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_bucket_file-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_content-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadata-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-file_metadatas-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-node-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-report-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-resource_type-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-resource_types-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-static_file_content-/*/-percentage`
 - `puppetlabs.<PRIMARY_HOSTNAME>.http.puppet-v3-status-/*/-percentage`
- `puppetlabs.<PRIMARY_HOSTNAME>.http.total-requests`: The total requests handled by Puppet Server.

JRuby metrics:

- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-count`: The number of successfully borrowed JRubies.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-retry-count`: The number of attempts to borrow a JRuby that must be retried.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.borrow-timeout-count`: The number of attempts to borrow a JRuby that resulted in a timeout.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.request-count`: The number of requested JRubies.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.return-count`: The number of JRubies successfully returned to the pool.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.num-free-jrubies`: The number of free JRuby instances. If this number is often 0, more requests are coming in than the server has available JRuby instances. To alleviate this, increase the number of JRuby instances on the Server or add additional compilers.
- `puppetlabs.<PRIMARY_HOSTNAME>.jruby.num-jrubies`: The total number of JRuby instances on the server, governed by the `max-active-instances` setting. See [Tuning JRuby on Puppet Server](#) for details.

Other metrics

These metrics measure raw resource availability and capacity.

- `puppetlabs.<PRIMARY_HOSTNAME>.num-cpus`: The number of available CPUs on the server.
- `puppetlabs.<PRIMARY_HOSTNAME>.uptime`: The Puppet Server process's uptime.

- Total, heap, and non-heap memory that's committed (committed), initialized (init), and used (used), and the maximum amount of memory that can be used (max).
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.total.committed`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.total.init`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.total.used`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.total.max`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.committed`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.init`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.used`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.heap.max`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.committed`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.init`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.used`
 - `puppetlabs.<PRIMARY_HOSTNAME>.memory.non-heap.max`

Related information

[Modifying exported metrics](#) on page 296

In addition to the default metrics, you can also export metrics measuring specific environments and nodes managed by Puppet Server.

Modifying exported metrics

In addition to the default metrics, you can also export metrics measuring specific environments and nodes managed by Puppet Server.

The `$metrics_puppetserver_metrics_allowed` class parameter in the `puppet_enterprise::profile::master` class takes an array of metrics as strings. To export additional metrics, add them to this array.

Optional metrics include:

- `compiler.compile.<ENVIRONMENT>` and `compiler.compile.<ENVIRONMENT>.<NODE-NAME>`, and all statistical fields suffixed to these (such as `compiler.compile.<ENVIRONMENT>.mean`).
- `compiler.compile.evaluate_resources.<RESOURCE>`: Time spent evaluating a specific resource during catalog compilation.

Omit the `puppetlabs.<MASTER-HOSTNAME>` prefix and field suffixes (such as `.count` or `.mean`) from metrics. Instead, suffix the environment or node name as a field to the metric.

1. For example, to track the compilation time for the production environment, add `compiler.compile.production` to the `metrics-allowed` list.
2. To track only the `my.node.localdomain` node in the production environment, add `compiler.compile.production.my.node.localdomain` to the `metrics-allowed` list.

Status API

The status API allows you to check the health of PE components and services. It can be useful for automated monitoring of your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

You can check the overall health of the console service, as well as the health of the individual services within the console service:

- Activity service
- Classifier service
- PuppetDB
- Puppet Server
- Role-based access control (RBAC) service

- Code Manager service

The endpoints provide overview health information in an overall healthy/error/unknown status field, and fine-detail information such as the availability of the database, the health of other required services, or connectivity to the primary server.

- [Authenticating to the status API](#) on page 297

Token-based authentication is not required to access the status API. You can choose to authenticate requests by using allowed certificates, or can access the API without authentication via HTTP.

- [Forming requests to the status API](#) on page 297

The HTTPS status endpoints are available on the console services API server, which uses port 4433 by default.

- [JSON endpoints](#) on page 298

These two endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

- [Activity service plaintext endpoints](#) on page 300

The activity service plaintext endpoints are designed for load balancers that don't support any kind of JSON parsing or parameter setting. They return simple string bodies (either the state of the service in question or a simple error message) and a status code relevant to the status result.

- [Metrics endpoints](#) on page 301

Puppet Server is capable of tracking advanced metrics to give you additional insight into its performance and health.

- [The metrics API](#) on page 307

Puppet Enterprise includes an optional, enabled-by-default web endpoint for Java Management Extension (JMX) metrics, namely managed beans (MBeans).

Related information

[API index](#) on page 26

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

Authenticating to the status API

Token-based authentication is not required to access the status API. You can choose to authenticate requests by using allowed certificates, or can access the API without authentication via HTTP.

You can authenticate requests using a certificate listed in RBAC's certificate allowlist, located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. The certificate allowlist is a simple, flat file consisting of certnames that match the host, for example:

```
node1.example
node2.example
node3.example
```

Note: If you edit the certificate allowlist, you must reload the `pe-console-services` service (run `sudo service pe-console-services reload`) for your changes to take effect.

The status API's endpoints can be served over HTTP, which does not require any authentication. This is disabled by default.

Tip: To use HTTP, locate the **PE Console** node group in the console, and in the `puppet_enterprise::profile::console` class, set `console_services_plaintext_status_enabled` to `true`.

Forming requests to the status API

The HTTPS status endpoints are available on the console services API server, which uses port 4433 by default.

The path prefix is `/status`, so, for example, the URL to get the statuses for all services as JSON is `https://<DNS NAME OF YOUR CONSOLE HOST>:4433/status/v1/services`.

To access that URL using curl commands, run:

```
cert="$(puppet config print hostcert)"
```

```
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/status/v1/services"

curl --cert "$cert" --cacert "$cacert" --key "$key" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

If enabled, the HTTP status endpoints are available on port 8123. See [Authenticating to the status API](#) on page 297 to enable HTTP.

Tip: To change the port, locate the **PE Console** node group in the console, and in the **puppet_enterprise::profile::console** class, set the **console_services_plaintext_status_port** parameter to your desired port number.

JSON endpoints

These two endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

GET /status/v1/services

Use the `/services` endpoint to retrieve the statuses of all PE services.

The content type for this endpoint is `application/json; charset=utf-8`.

Query parameters

The request accepts the following parameters:

Parameter	Value
level	How thorough of a check to run. Set to <code>critical</code> , <code>debug</code> , or <code>info</code> (default).
timeout	Specified in seconds; defaults to 30.

Response format

The response is a JSON object that lists details about the services, using the following keys:

Key	Definition
service_version	Package version of the JAR file containing a given service.
service_status_version	The version of the API used to report the status of the service.
detail_level	Can be <code>critical</code> , <code>debug</code> , or <code>info</code> .
state	Can be <code>running</code> , <code>error</code> , or <code>unknown</code> .
status	An object with the service's status details. Usually only relevant for error and unknown states.
active_alerts	An array of objects containing severity and a message about your replication from pglogical if you have replication enabled; otherwise, it's an empty array.

For example:

```
{ "rbac-service": { "service_version": "1.8.11-SNAPSHOT",
                  "service_status_version": 1,
                  "detail_level": "info",
```

```

    "state": "running",
    "status": {
      "activity_up": true,
      "db_up": true,
      "db_pool": { "state": "ready" },
      "replication": { "mode": "none", "status": "none" }
    },
    "active_alerts": [],
    "service_name": "rbac-service"
  }

  "classifier-service": { "service_version": "1.8.11-SNAPSHOT",
    "service_status_version": 1,
    "detail_level": "info",
    "state": "running",
    "status": {
      "activity_up": true,
      "db_up": true,
      "db_pool": { "state": "ready" },
      "replication": { "mode": "none", "status": "none" }
    },
    "active_alerts": [],
    "service_name": "classifier-service"
  }

```

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error
- 400 if a level parameter is set but is invalid (not critical, debug, or info)

GET /status/v1/services/<SERVICE NAME>

Use the /services/<SERVICE NAME> endpoint to retrieve the status of a particular PE service.

The content type for this endpoint is `application/json; charset=utf-8`.

Query parameters

The request accepts the following parameters:

Parameter	Value
level	How thorough of a check to run. Set to critical, debug, or info (default).
timeout	Specified in seconds; defaults to 30.

Response format

The response is a JSON object that lists details about the service, using the following keys:

Key	Definition
service_version	Package version of the JAR file containing a given service.
service_status_version	The version of the API used to report the status of the service.

Key	Definition
detail_level	Can be critical, debug, or info.
state	Can be running, error, or unknown.
status	An object with the service's status details. Usually only relevant for error and unknown states.
active_alerts	An array of objects containing severity and a message about your replication from pglogical if you have replication enabled; otherwise, it's an empty array.

For example:

```
{ "rbac-service": { "service_version": "1.8.11-SNAPSHOT",
  "service_status_version": 1,
  "detail_level": "info",
  "state": "running",
  "status": {
    "activity_up": true,
    "db_up": true,
    "db_pool": { "state": "ready" },
    "replication": { "mode": "none", "status": "none" }
  },
  "active_alerts": [],
}
```

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error
- 400 if a level parameter is set but is invalid (not critical, debug, or info)
- 404 if no service named <SERVICE NAME> is found

Activity service plaintext endpoints

The activity service plaintext endpoints are designed for load balancers that don't support any kind of JSON parsing or parameter setting. They return simple string bodies (either the state of the service in question or a simple error message) and a status code relevant to the status result.

GET /status/v1/simple

The `/status/v1/simple` returns a status that reflects all services the status service knows about.

The content type for this endpoint is `text/plain; charset=utf-8`.

Query parameters

No parameters are supported. Defaults to using the critical status level.

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error

Possible responses

The endpoint returns a status that reflects all services it knows about. It decides on what status to report using the following logic:

- `running` if and only if all services are running
- `error` if any service reports an error
- `unknown` if any service reports an unknown and no services report an error

GET /status/v1/simple/<SERVICE NAME>

The `/status/v1/simple/<SERVICE NAME>` endpoint returns the plaintext status of the specified service, such as `rbac-service` or `classifier-service`.

The content type for this endpoints is `text/plain; charset=utf-8`.

Query parameters

No parameters are supported. Defaults to using the `critical` status level.

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of `running`
- 503 if any service's status is `unknown` or `error`
- 404 if no service named `<SERVICE NAME>` is found

Possible responses

The endpoint returns a status that reflects all services it knows about. It decides on what status to report using the following logic:

- `running` if and only if all services are running
- `error` if any service reports an error
- `unknown` if any service reports an unknown and no services report an error
- `not found: <SERVICENAME>` if any service can't be found

Metrics endpoints

Puppet Server is capable of tracking advanced metrics to give you additional insight into its performance and health.

The HTTPS metrics endpoints are available on port 8140 of the primary server:

```
uri="https://$(puppet config print server):8140/status/v1/services"
curl --insecure "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Note: These API endpoints are a tech preview. The metrics described here are returned only when passing the `level=debug` URL parameter, and the structure of the returned data might change in future versions.

These metrics fall into three categories:

- JRuby metrics (`/status/v1/services/pe-jruby-metrics`)
- HTTP route metrics (`/status/v1/services/pe-master`)
- Catalog compilation profiler metrics (`/status/v1/services/pe-puppet-profiler`)

All of these metrics reflect data for the lifetime of the current Puppet Server process and reset whenever the service is restarted. Any time-related metrics report milliseconds unless otherwise noted.

Like the standard status endpoints, the metrics endpoints return machine-consumable information about running services. This JSON response includes the same keys returned by a standard status endpoint request (see JSON endpoints). Each endpoint also returns additional keys in an `experimental` section.

GET /status/v1/services/pe-jruby-metrics

The `/status/v1/services/pe-jruby-metrics` endpoint returns JSON containing information about the JRuby pools from which Puppet Server fulfills agent requests.

You must query it at port 8140 and append the `level=debug` URL parameter.

Query parameters

No parameters are supported. Defaults to using the `critical` status level.

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error

Response keys

The metrics are returned in two subsections of the `experimental` section: `jruby-pool-lock-status` and `metrics`.

The response's `experimental/jruby-pool-lock-status` section contains the following keys:

Key	Definition
<code>current-state</code>	The state of the JRuby pool lock, which should be either <code>:not-in-use</code> (unlocked), <code>:requested</code> (waiting for lock), or <code>:acquired</code> (locked).
<code>last-change-time</code>	The date and time of the last <code>current-state</code> update, formatted as an ISO 8601 combined date and time in UTC.

The response's `experimental/metrics` section contains the following keys:

Key	Definition
<code>average-borrow-time</code>	The average amount of time a JRuby instance spends handling requests, calculated by dividing the total duration in milliseconds of the <code>borrowed-instances</code> value by the <code>borrow-count</code> value.
<code>average-free-jrubies</code>	The average number of JRuby instances that are not in use over the Puppet Server process's lifetime.
<code>average-lock-held-time</code>	The average time the JRuby pool held a lock, starting when the value of <code>jruby-pool-lock-status/current-state</code> changed to <code>:acquired</code> . This time mostly represents file sync syncing code into the live codedir, and is calculated by dividing the total length of time that Puppet Server held the lock by the value of <code>num-pool-locks</code> .

Key	Definition
<code>average-lock-wait-time</code>	The average time Puppet Server spent waiting to lock the JRuby pool, starting when the value of <code>jruby-pool-lock-status/current-state</code> changed to <code>:requested</code>). This time mostly represents how long Puppet Server takes to fulfill agent requests, and is calculated by dividing the total length of time that Puppet Server waits for locks by the value of <code>num-pool-locks</code> .
<code>average-requested-jrubies</code>	The average number of requests waiting on an available JRuby instance over the Puppet Server process's lifetime.
<code>average-wait-time</code>	The average time Puppet Server spends waiting to reserve an instance from the JRuby pool, calculated by dividing the total duration in milliseconds of requested-instances by the <code>requested-count</code> value.
<code>borrow-count</code>	The total number of JRuby instances that have been used.
<code>borrow-retry-count</code>	The total number of times that a borrow attempt failed and was retried, such as when the JRuby pool is flushed while a borrow attempt is pending.
<code>borrow-timeout-count</code>	The number of requests that were not served because they timed out while waiting for a JRuby instance.
<code>borrowed-instances</code>	<p>A list of the JRuby instances currently in use, each reporting:</p> <ul style="list-style-type: none"> <code>duration-millis</code>: The length of time that the instance has been running. <code>reason/request</code>: A hash of details about the request being served. <ul style="list-style-type: none"> <code>request-method</code>: The HTTP request method, such as POST, GET, PUT, or DELETE. <code>route-id</code>: The route being served. For routing metrics, see the HTTP metrics endpoint. <code>uri</code>: The request's full URI. <code>time</code>: The time (in milliseconds since the Unix epoch) when the JRuby instance was borrowed.
<code>num-free-jrubies</code>	The number of JRuby instances in the pool that are ready to be used.
<code>num-jrubies</code>	The total number of JRuby instances.
<code>num-pool-locks</code>	The total number of times the JRuby pools have been locked.
<code>requested-count</code>	The number of JRuby instances borrowed, waiting, or that have timed out.

Key	Definition
requested-instances	<p>A list of the requests waiting to be served, each reporting:</p> <ul style="list-style-type: none"> • <code>duration-millis</code>: The length of time the request has waited. • <code>reason/request</code>: A hash of details about the waiting request. <ul style="list-style-type: none"> • <code>request-method</code>: The HTTP request method, such as POST, GET, PUT, or DELETE. • <code>route-id</code>: The route being served. For routing metrics, see the HTTP metrics endpoint. • <code>uri</code>: The request's full URI. • <code>time</code>: The time (in milliseconds since the Unix epoch) when Puppet Server received the request.
return-count	The total number of JRuby instances that have been used.

For example:

```
"pe-jruby-metrics": {
  "detail_level": "debug",
  "service_status_version": 1,
  "service_version": "2.2.22",
  "state": "running",
  "status": {
    "experimental": {
      "jruby-pool-lock-status": {
        "current-state": ":not-in-use",
        "last-change-time": "2015-12-03T18:59:12.157Z"
      },
      "metrics": {
        "average-borrow-time": 292,
        "average-free-jrubies": 0.4716243097301104,
        "average-lock-held-time": 1451,
        "average-lock-wait-time": 0,
        "average-requested-jrubies": 0.21324752542875958,
        "average-wait-time": 156,
        "borrow-count": 639,
        "borrow-retry-count": 0,
        "borrow-timeout-count": 0,
        "borrowed-instances": [
          {
            "duration-millis": 3972,
            "reason": {
              "request": {
                "request-method": "post",
                "route-id": "puppet-v3-catalog-/*/",
                "uri": "/puppet/v3/catalog/"
              }
            }
          }
        ]
      }
    },
    "time": 1448478371406
  }
}
```



```

    ],
    "num-free-jrubies": 0,
    "num-jrubies": 1,
    "num-pool-locks": 2849,
    "requested-count": 640,
    "requested-instances": [
      {
        "duration-millis": 3663,
        "reason": {
          "request": {
            "request-method": "put",
            "route-id": "puppet-v3-report-*/",
            "uri": "/puppet/v3/report/"
          }
        }
      },
      {
        "time": 1448478371715
      }
    ],
    "return-count": 638
  }
}
}
}
}
}
hostname.example.com"

```

GET /status/v1/services/pe-master

The `/status/v1/services/pe-master` endpoint returns JSON containing information about the routes that agents use to connect to this server.

You must query it at port 8140 and append the `level=debug` URL parameter.

Query parameters

No parameters are supported. Defaults to using the `critical` status level.

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of running
- 503 if any service's status is unknown or error

Response keys

The response's `experimental/http-metrics` section contains a list of routes, each containing the following keys:

Key	Definition
aggregate	The total time Puppet Server spent processing requests for this route.
count	The total number of requests Puppet Server processed for this route.
mean	The average time Puppet Server spent on each request for this route, calculated by dividing the aggregate value by the count.
route-id	The route being served. The request returns a route with the special <code>route-id</code> of "total", which represents the aggregate data for all requests along all routes.

Routes for newer versions of Puppet Enterprise and newer agents are prefixed with `puppet-v3`, while Puppet Enterprise 3 agents' routes are not. For example, a PE 2017.3 `route-id` might be `puppet-v3-report-/*`, while the equivalent PE 3 agent's `route-id` is `:environment-report-/*`.

For example:

```
"pe-master": {
  {...},
  "status": {
    "experimental": {
      "http-metrics": [
        {
          "aggregate": 70668,
          "count": 234,
          "mean": 302,
          "route-id": "total"
        },
        {
          "aggregate": 28613,
          "count": 13,
          "mean": 2201,
          "route-id": "puppet-v3-catalog-/*/"
        },
        {...}
      ]
    }
  }
}
```

GET /status/v1/services/pe-puppet-profiler

The `/status/v1/services/pe-puppet-profiler` endpoint returns JSON containing statistics about catalog compilation. You can use this data to discover which functions or resources are consuming the most resources or are most frequently used.

You must query it at port 8140 and append the `level=debug` URL parameter.

The Puppet Server profiler is enabled by default, but if it has been disabled, this endpoint's metrics are not available. Instead, the endpoint returns the same keys returned by a standard status endpoint request and an empty `status` key.

Query parameters

No parameters are supported. Defaults to using the `critical` status level.

Response codes

The server uses the following response codes:

- 200 if and only if all services report a status of `running`
- 503 if any service's status is `unknown` or `error`

Response keys

If the profiler is enabled, the response returns two subsections in the `experimental` section:

- `experimental/function-metrics`, containing statistics about functions evaluated by Puppet Server when compiling catalogs.
- `experimental/resource-metrics`, containing statistics about resources declared in manifests compiled by Puppet Server.

Each function measured in the `function-metrics` section also has a **function** key containing the function's name, and each resource measured in the `resource-metrics` section has a **resource** key containing the resource's name.

The two sections otherwise share these keys:

Key	Definition
aggregate	The total time spent handling this function call or resource during catalog compilation.
count	The number of times Puppet Server has called the function or instantiated the resource during catalog compilation.
mean	The average time spent handling this function call or resource during catalog compilation, calculated by dividing the aggregate value by the count.

For example:

```
"pe-puppet-profiler": {
  {...},
  "status": {
    "experimental": {
      "function-metrics": [
        {
          "aggregate": 1628,
          "count": 407,
          "function": "include",
          "mean": 4
        },
        {...},
      ]
      "resource-metrics": [
        {
          "aggregate": 3535,
          "count": 5,
          "mean": 707,
          "resource": "Class[Puppet_enterprise::Profile::Console]"
        },
        {...},
      ]
    }
  }
}
```

The metrics API

Puppet Enterprise includes an optional, enabled-by-default web endpoint for Java Management Extension (JMX) metrics, namely managed beans (MBeans).

These endpoints include:

- GET /metrics/v1/mbeans
- POST /metrics/v1/mbeans
- GET /metrics/v1/mbeans/<name>

Note: These API endpoints are a tech preview. The metrics described here are returned only when passing the `level=debug` URL parameter, and the structure of the returned data might change in future versions. To disable this endpoint, set `puppet_enterprise::master::puppetserver::metrics_webservice_enabled: false` in `Hiera`.

GET /metrics/v1/mbeans

The GET `/metrics/v1/mbeans` endpoint lists available MBeans.

Response keys

- The key is the name of a valid MBean.
- The value is a URI to use when requesting that MBean's attributes.

POST /metrics/v1/mbeans

The POST `/metrics/v1/mbeans` endpoint retrieves requested MBean metrics.

Query parameters

The query doesn't require any parameters, but the request body must contain a JSON object whose values are metric names, or a JSON array of metric names, or a JSON string containing a single metric's name.

For a list of metric names, make a GET request to `/metrics/v1/mbeans`.

Response keys

The response format, though always JSON, depends on the request format:

- Requests with a JSON object return a JSON object where the values of the original object are transformed into the Mbeans' attributes for the metric names.
- Requests with a JSON array return a JSON array where the items of the original array are transformed into the Mbeans' attributes for the metric names.
- Requests with a JSON string return the a JSON object of the Mbean's attributes for the given metric name.

GET /metrics/v1/mbeans/<name>

The GET `/metrics/v1/mbeans/<name>` endpoint reports on a single metric.

Query parameters

The query doesn't require any parameters, but the endpoint itself must correspond to one of the metrics returned by a GET request to `/metrics/v1/mbeans`.

Response keys

The endpoint's responses contain a JSON object mapping strings to values. The keys and values returned in the response vary based on the specified metric.

For example:

Use `curl` from localhost to request data on MBean memory usage:

```
curl 'http://localhost:8080/metrics/v1/mbeans/java.lang:type=Memory'
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The response contains a JSON object representing the data:

```
{
  "ObjectPendingFinalizationCount" : 0,
  "HeapMemoryUsage" : {
    "committed" : 807403520,
    "init" : 268435456,
    "max" : 3817865216,
    "used" : 129257096
  },
  "NonHeapMemoryUsage" : {
    "committed" : 85590016,
```

```

    "init" : 24576000,
    "max" : 184549376,
    "used" : 85364904
  },
  "Verbose" : false,
  "ObjectName" : "java.lang:type=Memory"
}

```

Managing nodes

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

- [Adding and removing agent nodes](#) on page 310

After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.

- [Adding and removing agentless nodes](#) on page 312

Using the inventory, you can manage nodes, including devices such as network switches or firewalls, without installing the Puppet agent on them. The inventory stores node and device information securely.

- [How nodes are counted](#) on page 315

Your *node count* is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your *bursting limit*. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

- [Running Puppet on nodes](#) on page 316

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

- [Grouping and classifying nodes](#) on page 318

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

- [Making changes to node groups](#) on page 326

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

- [Environment-based testing](#) on page 328

An environment-based testing workflow is an effective approach for testing new code before pushing it to production.

- [Preconfigured node groups](#) on page 330

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

- [Managing Windows nodes](#) on page 334

You can use PE to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks using Forge modules.

- [Designing system configs: roles and profiles](#) on page 360

Your typical goal with Puppet is to build complete system configurations, which manage all of the software, services, and configuration that you care about on a given system. The roles and profiles method can help keep complexity under control and make your code more reusable, reconfigurable, and refactorable.

- [Node classifier API v1](#) on page 387

These are the endpoints for the node classifier v1 API.

- [Node classifier API v2](#) on page 436

These are the endpoints for the node classifier v2 API.

- [Node inventory API](#) on page 439

These are the endpoints for the node inventory v1 API.

Adding and removing agent nodes

After you install a Puppet agent on a node, accept its certificate signing request and begin managing it with Puppet Enterprise (PE). Or remove nodes that you no longer need.

Managing certificate signing requests

When you install a Puppet agent on a node, the agent automatically submits a certificate signing request (CSR) to the primary server. You must accept this request to bring before the node under PE management can be added your deployment. This allows Puppet to run on the node and enforce your configuration, which in turn adds node information to PuppetDB and makes the node available throughout the console.

You can approve certificate requests from the PE console or the command line. If DNS altnames are set up for agent nodes, you must approve the CSRs on use the command line interface .

Note: Specific user permissions are required to manage certificate requests:

- To accept or reject CSRs in the console or on the command line, you need the permission **Certificate requests: Accept and reject**.
- To manage certificate requests in the console, you also need the permission **Console: View**.

Related information

[Installing agents](#) on page 112

You can install Puppet Enterprise agents on *nix, Windows, and macOS.

Managing certificate signing requests in the console

A certificate signing request appears in the console on the **Certificates** page in the **Unsigned certificates** tab after you add an agent node to inventory. Accept or reject submitted requests individually or in a batch.

- To manage requests individually, click **Accept** or **Reject**.
- To manage the entire list of requests, click **Accept All** or **Reject All**. Nodes are processed in batches. If you close the browser window or navigate to another website while processing is in progress, only the current batch is processed.

After you accept the certificate signing request, the node appears in the console after the next Puppet run. To make a node available immediately after you approve the request, run Puppet on demand.

Related information

[Running Puppet on demand](#) on page 476

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

Managing certificate signing requests on the command line

You can view, approve, and reject node requests using the command line.

To view pending node requests on the command line:

```
$ sudo puppetserver ca list
```

To sign a pending request:

```
$ sudo puppetserver ca sign --certname <NAME>
```

Note: You can use the Puppet Server CA CLI to sign certificates with altnames or auth extensions by default.

Remove agent nodes

If you no longer wish to manage an agent node, you can remove it and make its license available for another node.

Purging a node:

- Removes the node from PuppetDB.
- Deletes the primary server's information cache for the node.
- Makes the license available for another node.
- Makes the hostname available for another node.

Note: Removing a node doesn't uninstall the agent from the node.

1. On the agent node, stop the agent service: `service puppet stop`
2. On the primary server, purge the node: `puppet node purge <CERTNAME>`

The node's certificate is revoked, the certificate revocation list (CRL) is updated, and the node is removed from PuppetDB and the console. The license is now available for another node. The node can't check in or re-register with PuppetDB on the next Puppet run.

3. If you have compilers, run Puppet on them: `puppet agent -t`

The updated CRL is managed by Puppet and distributed to compilers.

4. Optional: If the node you're removing was pinned to any node groups, you must manually unpin it from individual node groups or from all node groups using the `unpin-from-all` command endpoint.

Related information

[Uninstall infrastructure nodes](#) on page 142

The `puppet-enterprise-uninstaller` script is installed on the primary server. In order to uninstall, you must run the uninstaller on each infrastructure node.

[POST /v1/commands/unpin-from-all](#) on page 421

Use the `/v1/commands/unpin-from-all` to unpin specified nodes from all groups they're pinned to. Nodes that are dynamically classified using rules aren't affected.

Uninstall *nix agents

The *nix agent package includes an uninstall script, which you can use when you're ready to retire a node.

1. On the agent node, run the uninstall script: `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`
2. Follow prompts to uninstall.
3. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean <AGENT CERT NAME>`

Uninstall Windows agents

To uninstall the agent from a Windows node, use the Windows **Add or Remove Programs** interface, or uninstall from the command line.

Uninstalling the agent removes the Puppet program directory, the agent service, and all related registry keys. The data directory remains intact, including all SSL keys. To completely remove Puppet from the system, manually delete the data directory.

1. Use the Windows **Add or Remove Programs** interface to remove the agent.

Alternatively, you can uninstall from the command line if you have the original .msi file or know the product code of the installed MSI, for example: `msiexec /qn /norestart /x [puppet.msi|<PRODUCT_CODE>]`

2. (Optional) If you plan to reinstall on the node at a later date, remove the agent certificate for the agent from the primary server: `puppetserver ca clean <AGENT CERT NAME>`

Adding and removing agentless nodes

Using the inventory, you can manage nodes, including devices such as network switches or firewalls, without installing the Puppet agent on them. The inventory stores node and device information securely.

The inventory connects to agentless nodes through SSH or WinRM remote connections. The inventory uses transport definitions from installed device modules to connect to devices that can't have an agent installed on them.

After you add credentials to the inventory, authorized users can run tasks on these nodes and devices without re-entering credentials. On the **Tasks** page, these nodes and devices appear in the same list of targets as those that have agents installed.

Add agentless nodes to the inventory

Add nodes over SSH or WinRM that will not or cannot have the Puppet agent installed to the inventory so you can run tasks on them.

Before you begin

Add classes to the **PE Master** node group for each agent platform used in your environment. For example, `pe_repo::platform::el_7_x86_64`.

Make sure you have the permission **Nodes: Add and delete connection information from inventory service**.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect over SSH or WinRM**.

3. Select a transport method.
 - **SSH** for *nix targets
 - **WinRM** for Windows targets
4. Enter target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
5. Optional: Select additional target options. For example, to add a target port number, select **Target Port** from the drop-down list, enter the number, and click **Add**. For details, see [Transport configuration options](#) on page 313.
6. Click **Add nodes**.

After the nodes have been added to the inventory, they are added to PuppetDB, and you can view them from the **Nodes** page. Nodes in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory node's connection options, or to remove the node from inventory, go to the **Connections** tab on the node's details page.

Transport configuration options

A list of transport configuration options for SSH and WinRM transports.

Target options for SSH transport

Option	Definition
Target port	Connection port. Default is 22.
Connection time-out in seconds	The length of time PE should wait when establishing connections.
Run as another user	After login, the user name to use for running commands.
Temporary directory	The directory to use when uploading temporary files to the target.
Sudo password	Password to use when changing users via <code>run-as</code> .
Process request as tty	Enable text terminal allocation.

Target options for WinRM transport

Option	Definition
Target port	Connection port. Default is 5986, or 5985 if <code>ssl : false</code>
Connection time-out in seconds	The length of time PE should wait when establishing connections.
Temporary directory	The directory to use when uploading temporary files to the target.
Acceptable file extension	List of file extensions that are accepted for scripts or tasks. Scripts with these file extensions rely on the target node's file type association to run. For example, if Python is installed on the system, a <code>.py</code> script should run with <code>python.exe</code> . The extensions <code>.ps1</code> , <code>.rb</code> , and <code>.pp</code> are always allowed and run via hard-coded executables.

Add devices to the inventory

If you have installed modules for device transports in your production environment, you can add connections to those devices to your inventory. This lets you manage network devices such as switches and firewalls, and run Puppet and task jobs on them, just like other agentless nodes in your infrastructure.

Before you begin

Make sure you have the permission **Nodes: Add and delete connection information from inventory service**.

The connection details differ for each type of device transport, as defined in the module; see the device module's README for details.



CAUTION: This initial implementation of network device configuration management is limited in how many device connections it can handle. Managing more than 100 devices can cause performance issues on the primary server. We plan to gather feedback from this release and improve the scaling and performance capabilities. In the meantime, avoid impacting primary server performance by limiting the number of network device connections you make to 100.

1. In the console, on the **Nodes** page, click **Add nodes**.
2. Click **Connect network devices**.
3. Select a device type from the list of device transports that you have installed as modules in your production environment.
4. Enter the device certname and other connection details, as defined in the transport module. Mandatory fields are marked with an asterisk. See the module README file if you need more details or examples specific to the transport.
5. Click **Add node**.

After devices have been added to the inventory, they are added to PuppetDB, and you can view them from the **Nodes** page. Devices in the inventory can be added to an inventory node list when you set up a job to run tasks. To review each inventory device's connection options, or to remove the device from inventory, go to the **Connections** tab on the device's node details page.

Remove agentless nodes and devices from the inventory

Remove an agentless node or device connection from the inventory from the **Connections** tab on its details page.

Before you begin

Make sure you have the permission **Nodes: Add and delete connection information from inventory service**.

1. On the **Status** or **Nodes** page, find the node or device whose connection you want to remove, and click its name to open its details page.
2. Click **Connections**.
3. Click **Remove connection**. The exact name of the link varies depending on the connection type: **Remove SSH Connection**, **Remove WinRM connection**, or similar.
4. Confirm that you want to remove the connection.

When you remove a node or device connection from the inventory, PuppetDB marks it as expired after the standard node time-to-live (`node-ttl`) and then purges the node when it reaches its node-purge time-to-live limit (`node-purge-ttl`). At this point the node no longer appears in the console, and the node's license is available for use.

Tip: For more information about `node-ttl` and `node-purge-ttl` settings, see the PE docs for [database settings](#).

Related information

[Node inventory API](#) on page 439

These are the endpoints for the node inventory v1 API.

How nodes are counted

Your *node count* is the number of nodes in your inventory. Your license limits you to a certain number of active nodes before you hit your *bursting limit*. If you hit your bursting limit on four days during a month, you must purchase a license for more nodes or remove some nodes from your inventory.

Note: *Node* in this context includes agent nodes, agentless nodes, primary servers, compilers, nodes running in `noop` mode, and nodes that have been purged but had prior activity within the same calendar month.

Nodes included in the node count

The following nodes are included in your node count:

- Nodes with a report in PuppetDB during the calendar month.
- Nodes that have executed a Puppet run, task, or plan in the orchestrator, even if they do not have a report during the calendar month.

Nodes not included in the node count

The following nodes are not included in your node count:

- Nodes that are in the inventory service but are not used with Puppet runs, tasks, or plans.
- Nodes that have been purged and have no reports or activity within the calendar month.

Reaching the bursting limit

When you go above your node license's count limit, you enter what is called the *bursting limit*. The bursting limit lets you to exceed the number of nodes allowed under your license and enter a new threshold without extra charge. You are allowed to use your bursting limit on four consecutive or non-consecutive days per calendar month. Any higher usage beyond the four days will require you to either purge nodes or buy a license for more nodes.

The amount of time within your bursting limit does not matter for it to be counted as one day. For example, assuming you are licensed to use 1000 nodes and your bursting limit is 2000 nodes:

- If you use 1200 nodes for two hours one day, you have three days left to use your bursting limit that month.
- If you use 1900 nodes for 23 hours one day, you have three days left to use your bursting limit that month.
- If you use 1500 nodes for one hour each day for four days, you must contact your Puppet representative to buy a license for more nodes or purge some of your nodes until the next calendar month.

When nodes are counted

PE tracks node counts daily from 12:00 midnight UTC to 12:00 midnight UTC.

The same time is used for the calendar month. For example, the month of September would include activity from 12:00 midnight UTC 01 September until 12:00 midnight UTC 01 October. After that point, the bursting limit restarts with a fresh four days in October.

Viewing your node count

View your daily node count in the console by navigating to the **License** page and scrolling to the **Calendar month usage** section. This section also contains information about your subscription expiration date and license warnings, such as your license being expired or out of compliance.

To see daily node usage information on the command line, use the [Puppet orchestrator API: usage endpoint](#) on page 607.

Removing nodes

If you have unused nodes cluttering your inventory and are concerned about reaching your limit, read about removing them in [Adding and removing agent nodes](#) on page 310 and [Adding and removing agentless nodes](#) on page 312.

Related information

[Purchasing and installing a license key](#) on page 111

Your license must support the number of nodes that you want to manage with Puppet Enterprise.

Running Puppet on nodes

Puppet automatically attempts to run on each of your nodes every 30 minutes. To trigger a Puppet run outside of the default 30-minute interval, you can manually run Puppet.

In a Puppet run, the primary server and agent nodes perform the following actions:

1. The agent node sends facts to the primary server and requests a catalog.
2. The primary server compiles and returns the agent's catalog.
3. The agent applies the catalog by checking each resource the catalog describes. If it finds any resources that are not in the desired state, it makes the necessary changes.

Note: Puppet run behavior differs slightly if static catalogs are enabled.

Related information

[Static catalogs](#) on page 182

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. A primary server typically compiles a catalog from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata that specifies the desired state of any file resources containing `source` attributes pointing to puppet:/// locations on a node.

Running Puppet with the orchestrator

The Puppet orchestrator is a set of interactive tools used to deploy configuration changes when and how you want them. You can use the orchestrator to run Puppet from the console, command line, or API.

You can use the orchestrator to enforce change based on a:

- selection of nodes – from the console or the command line:

```
puppet job run --nodes <COMMA-SEPARATED LIST OF NODE NAMES>
```

- PQL nodes query – from the console or the command line, for example:

```
puppet job run --query 'nodes[certname] { facts {name = "operatingsystem"
and value = "Debian" } }'
```

- application or application instance - from the command line.

If you're putting together your own tools for running Puppet or want to enable CI workflows across your infrastructure, use the orchestrator API.

Related information

[Running Puppet on demand from the console](#) on page 476

When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

Running Puppet with SSH

Run Puppet with SSH from an agent node.

SSH into the node and run `puppet agent --test` or `puppet agent -t`.

Running Puppet from the console

In the console, you can run Puppet from the node detail page for nodes that have an agent connection.

Run options include:

- **No-op** – Simulates changes without actually enforcing a new catalog. Nodes with `noop = true` in their `puppet.conf` files always run in no-op mode.
- **Debug** – Prints all messages available for use in debugging.
- **Trace** – Prints stack traces on some errors.
- **Evaltrace** – Shows a breakdown of the time taken for each step in the run.

When the run completes, the console displays the node's run status.

Note: The **Run Puppet** button is not available if an agent does not have an active websocket session with the PCP broker, or if the node's connection method is SSH, WinRM, or a network device transport.

Related information

[Node run statuses](#) on page 270

The **Status** page displays the run status of each node following the most recent Puppet run. There are 10 possible run statuses.

Activity logging on console Puppet runs

When you initiate a Puppet run from the console, the Activity service logs the activity.

You can view activity on a single node by selecting the node, then clicking the **Activity** tab.

Alternatively, you can use the Activity Service API to retrieve activity information.

Related information

[Activity service API](#) on page 257

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

Troubleshooting Puppet run failures

Puppet creates a **View Report** link for most failed runs, which you can use to access events and logs.

This table shows some errors that can occur when you attempt to run Puppet, and suggestions for troubleshooting.

Error	Possible Cause
Changes could not be applied	Conflicting classes are a common cause. Check the log to get more detail.
Noop, changes could not be applied	Conflicting classes are a common cause. Check the log to get more detail.
Run already in progress	Occurs when a run is triggered in the command line or by another user, and you click Run .
Run request times out	Occurs if you click Run and the agent isn't available.
Report request times out	Occurs when the report is not successfully stored in PuppetDB after the run completes.
Invalid response (such as a 500 error)	Your Puppet code might be have incorrect formatting.
Button is disabled and a run is not allowed.	The user has permission, but the agent is not responding.

Grouping and classifying nodes

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

The main steps involved in classifying nodes are:

1. Create *node groups*.
2. Add nodes to groups, either manually or dynamically, with *rules*.
3. Assign classes to node groups.

Nodes can match the rules of many node groups. They receive classes, class parameters, and variables from all the node groups that they match.

How node group inheritance works

Node groups exist in a hierarchy of parent and child relationships. Nodes inherit classes, class parameters and variables, and rules from all ancestor groups.

- **Classes** – If an ancestor node group has a class, all descendent node groups also have the class.
- **Class parameters and variables** – Descendent node groups inherit class parameters and variables from ancestors unless a different value is set for the parameter or variable in the descendent node group.
- **Rules** – A node group can only match nodes that all of its ancestors also match. Specifying rules in a child node group is a way of narrowing down the nodes in the parent node group to apply classes to a specific subset of nodes.

Because nodes can match multiple node groups from separate hierarchical lineages, it's possible for two equal node groups to contribute conflicting values for variables and class parameters. Conflicting values cause a Puppet run on an agent to fail.

Tip: In the console, you can see how node groups are related on the **Node groups** page, which displays a hierarchical view of node groups. From the command line, you can use the group children endpoint to review group lineage.

Related information

[GET /v1/group-children/:id](#) on page 426

Use the `/v1/group-children/:id` endpoint to retrieve a specified group and its descendents.

Best practices for classifying node groups

To organize node groups, start with the high-level functional groups that reflect the business requirements of your organization, and work down to smaller segments within those groups.

For example, if a large portion of your infrastructure consists of web servers, create a node group called `web servers` and add any classes that need to be applied to all web servers.

Next, identify subsets of web servers that have common characteristics but differ from other subsets. For example, you might have production web servers and development web servers. So, create a `dev web` child node group under the `web servers` node group. Nodes that match the `dev web` node group get all of the classes in the parent node group in addition to the classes assigned to the `dev web` node group.

Create node groups

You can create node groups to assign either an environment or classification.

- **Environment node groups** assign environments to nodes, such as test, development, or production.
- **Classification node groups** assign classification data to nodes, including classes, parameters, and variables.

Create environment node groups

Create custom environment node groups so that you can target deployment of Puppet code.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the new node group and then click **Add**.

- **Parent name** – Select the top-level environment node group in your hierarchy. If you're using default environment node groups, this might be **Production environment** or **All environments**. Every environment node group you add must be a descendant of the top-level environment node group.
- **Group name** – Enter a name that describes the role of this environment node group, for example, Test environment.
- **Environment** – Select the environment that you want to assign to nodes that match this node group.
- **Environment group** – Select this option.

You can now add nodes to your environment node group to control which environment each node belongs to.

Create classification node groups

Create classification node groups to assign classification data to nodes.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the new node group and then click **Add**.

- **Parent name** – Select the name of the classification node group that you want to set as the parent to this node group. Classification node groups inherit classes, parameters, and variables from their parent node group. By default, the parent node group is the **All Nodes** node group.
- **Group name** – Enter a name that describes the role of this classification node group, for example, Web Servers.
- **Environment** – Specify an environment to limit the classes and parameters available for selection in this node group.

Note: Specifying an environment in a classification node group does not assign an environment to any nodes, as it does in an environment node group.
- **Environment group** – Do not select this option.

You can now add nodes to your classification node group dynamically or statically.

Add nodes to a node group

There are two ways to add nodes to a node group.

- Individually pin nodes to the node group (static)
- Create rules that match node facts (dynamic)

Statically add nodes to a node group

If you have a node that needs to be in a node group regardless of the rules specified for that node group, you can pin the node to the node group.

A pinned node remains in the node group until you manually remove it. Adding a pinned node essentially creates the rule `<the certname of your node> = <the certname>`, and includes this rule along with the other fact-based rules.

1. In the console, click **Node groups**, and then find the node group that you want to pin a node to and select it.
2. On the **Rules** tab, in the pinned nodes section, enter the certname of the node.
3. Click **Pin node**, and then commit changes.

Dynamically add nodes to a node group

Rules are the most powerful and scalable way to include nodes in a node group. You can create rules in a node group that are used to match node facts.

When nodes match the rules in a node group, they're classified with all of the classification data (classes, parameters, and variables) for the node group.

When nodes no longer match the rules of a node group, the classification data for that node group no longer applies to the node.

1. In the console, click **Node Groups**, and then find the node group that you want to add the rule to and select it.

2. On the **Rules** tab, specify rules for the fact, then click **Add rule**.
3. (Optional) Repeat step 2 as needed to add more rules.

Tip: If the node group includes multiple rules, be sure to specify whether **Nodes must match all rules** or **Nodes can match any rule**.

4. Commit changes.

Writing node group rules

To dynamically assign nodes to a group, you must specify rules based on node facts. Use this reference to fill out the **Rules** tab for node groups.

Option	Definition
Fact	<p>Specifies the fact used to match nodes.</p> <p>Select from the list of known facts, or enter part of a string to view fuzzy matches.</p> <p>To use structured or trusted facts, select the initial value from the dropdown list, then type the rest of the fact.</p> <ul style="list-style-type: none"> • To descend into a hash, use dots (".") to designate path segments, such as <code>os.release.major</code> or <code>trusted.certname</code>. • To specify an item in an array, surround the numerical index of the item in square brackets, such as <code>processors.models[0]</code> or <code>mountpoints./.options[0]</code>. • To identify path segments that contain dots, periods, dashes, spaces, or UTF characters, surround the segment with single or double quotes, such as <code>trusted.extensions."1.3.6.1.4.1.34380.1.2.1"</code>. • To use trusted extension short names, append the short name after a second dot, such as <code>trusted.extensions.pp_role</code>. <p>Tip: Structured and trusted facts don't provide type-ahead suggestions beyond the top-level name key, and the facts aren't verified when entered. After adding a rule for a structured or trusted fact, review the number of matching nodes to verify that the fact was entered correctly.</p>

Option	Definition
Operator	<p>Describes the relationship between the fact and value.</p> <p>Operators include:</p> <ul style="list-style-type: none"> • = — is • != — is not • ~ — matches regex • !~ — does not match regex • > — greater than • >= — greater than or equal to • < — less than • <= — less than or equal to <p>The numeric operators >, >=, <, and <= can be used only with facts that have a numeric value.</p> <p>To match highly specific node facts, use ~ or !~ with a regular expression for Value.</p>
Value	Specifies the value associated with the fact.

Using structured and trusted facts for node group rules

Structured facts group a set of related facts, whereas trusted facts are a specific type of structured fact.

Structured facts group a set of related facts in the form of a hash or array. For example, the structured fact `os` includes multiple independent facts about the operating system, including architecture, family, and release. In the console, when you view facts about a node, you can differentiate structured facts because they're surrounded by curly braces.

Trusted facts are a specific type of structured fact where the facts are immutable and extracted from a node's certificate. Because they can't be changed or overridden, trusted facts enhance security by verifying a node's identity before sending sensitive data in its catalog.

You can use structured and trusted facts in the console to dynamically add nodes to groups.

Note: If you're using trusted facts to specify certificate extensions, in order for nodes to match to rules correctly, you must use short names for Puppet [registered IDs](#) and numeric IDs for [private extensions](#). Numeric IDs are required for private extensions whether or not you specify a short name in the `custom_trusted_oid_mapping.yaml` file.

Declare classes

Classes are the blocks of Puppet code used to configure nodes and assign resources to them.

Before you begin

The class that you want to apply must exist in an installed module. You can download modules from the Forge or create your own module.

1. In the console, click **Node groups**, and then find the node group that you want to add the class to and select it.

2. On the **Classes** tab, select the class to add.

The **Add new class** field suggests classes that the primary server knows about and that are available in the environment set for the node group.

3. Click **Add class** and then commit changes.

Note: Classes don't appear in the class list until they're retrieved from the primary server and the environment cache is refreshed. By default, both of these actions occur every three minutes. To override the default refresh period and force the node classifier to retrieve the classes from the primary server immediately, click the **Refresh** button.

Enable data editing in the console

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hiera.yaml` file.

1. On your primary server, edit `/etc/puppetlabs/puppet/hiera.yaml` to add:

```
hierarchy:
- name: "Classifier Configuration Data"
  data_hash: classifier_data
```

Place any additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same `hierarchy` key, preferably below the `Classifier Configuration Data` entry.

Note: If you enable data editing in the console, add both **Set environment** and **Edit configuration data** to groups that set environment or modify class parameters in order for users to make changes.

2. If your environment is configured for disaster recovery, update `hiera.yaml` on your replica.

Define data used by node groups

The console offers multiple ways to specify data used in your manifests.

- **Configuration data** — Specify values through automatic parameter lookup.
- **Parameters** — Specify resource-style values used by a declared class.
- **Variables** — Specify values to make available in Puppet code as top-scope variables.

Set configuration data

Configuration data set in the console is used for automatic parameter lookup, the same way that Hieradata is used. Console configuration data takes precedence over Hieradata, but you can combine data from both sources to configure nodes.

Tip: In most cases, setting configuration data in Hieradata is the more scalable and consistent method, but there are some cases where the console is preferable. Use the console to set configuration data if:

- You want to override Hieradata. Data set in the console overrides Hieradata when configured as recommended.
- You want to give someone access to set or change data and they don't have the skill set to do it in Hieradata.
- You simply prefer the console user interface.

Important: If your installation includes a disaster recovery replica, make sure you've correctly enabled data editing in the console for both your primary server and replica.

1. In the console, click **Node groups**, then find the node group that you want to add configuration data to and select it.
2. On the **Configuration data** tab, specify a **Class** and select a **Parameter** to add.

You can select from existing classes and parameters in the node group's environment, or you can specify free-form values. Classes aren't validated, but any class you specify must be present in the node's catalog at runtime in order for the parameter value to be applied.

When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. Optional: Change the default parameter **Value**.

Related information

[Enable data editing in the console](#) on page 173

The ability to edit configuration data in the console is enabled by default in new installations. If you upgrade from an earlier version and didn't previously have configuration data enabled, you must manually enable classifier configuration data, because enabling requires edits to your `hieradata.yaml` file.

Set parameters

Parameters are declared resource-style, which means they can be used to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

1. In the console, click **Node groups**, and then find the node group that you want to add a parameter to and select it.
2. On the **Classes** tab, select the class you want to modify and the **Parameter** to add.

The **Parameter** drop-down list shows all of the parameters that are available for that class in the node group's environment. When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

3. (Optional) Change the default **Value**.

Set variables

Variables set in the console become top-scope variables available to all Puppet manifests.

1. In the console, click **Node groups**, and then find the node group that you want to set a variable for and select it.
2. On the **Variables** tab, enter options for the variable:
 - **Key** – Enter the name of the variable.
 - **Value** – Enter the value that you want to assign to the variable.
3. Click **Add variable**, and then commit changes.

Tips for specifying parameter and variable values

Parameters and variables can be structured as JSON. If they can't be parsed as JSON, they're treated as strings.

Parameters and variables can be specified using these data types and syntax:

- Strings (for example, `"centos"`)
 - Variable-style syntax, which interpolates the result of referencing a fact (for example, `"I live at $ipaddress."`)
 - Expression-style syntax, which interpolates the result of evaluating the embedded expression (for example, ``${os}release``)

Note: Strings must be double-quoted, because single quotes aren't valid JSON.

Tip: To enter a value in the console that contains a literal dollar sign, like a password hash — for example, `1nnkkFwEc$safFMXYaUVfKrDV4FLCm0/` — escape each dollar sign with a backslash to disable interpolation.

- Booleans (for example, `true` or `false`)
- Numbers (for example, `123`)
- Hashes (for example, `{ "a": 1 }`)

Note: Hashes must use colons rather than hash rockets.

- Arrays (for example, `["1", "2.3"]`)

Variable-style syntax

Variable-style syntax uses a dollar sign (\$) followed by a Puppet fact name.

Example: `"I live at $ipaddress"`

Variable-style syntax is interpolated as the value of the fact. For example, `$ipaddress` resolves to the value of the `ipaddress` fact.

Important: The endpoint variable `$pe_node_groups` cannot be interpolated when used as a classifier in class variable values.

Indexing cannot be used in variable-style syntax because the indices are treated as part of the string literal. For example, given the following fact: `processors => { "count" => 4, "physicalcount" => 1 }`, if you use variable-style syntax to specify `$processors[count]`, the value of the `processors` fact is interpolated but it is followed by a literal `"[count]"`. After interpolation, this example becomes `{ "count" => 4, "physicalcount" => 1 }[count]`.

Note: Do not use the `::` top-level scope indication because the console is not aware of Puppet variable scope.

Expression-style syntax

Use expression-style syntax when you need to index into a fact (`${$os[release]}`), refer to trusted facts (`"My name is ${trusted[certname]}"`), or delimit fact names from strings (`"My ${os} release"`).

The following is an example of using expression-style syntax to access the full release number of an operating system:

```
${$os"release"}
```

Expression-style syntax uses the following elements in order:

- an initial dollar sign and curly brace (`${}`)
- a legal Puppet fact name preceded by an optional dollar sign
- any number of index expressions (the quotations around indices are optional but are required if the index string contains spaces or square brackets)
- a closing curly brace (`}`)

Indices in expression-style syntax can be used to access individual fields of structured facts, or to refer to trusted facts. Use strings in an index if you want to access the keys of a hashmap. If you want to access a particular item or character in an array or string based on the order in which it is listed, you can use an integer (zero-indexed).

Examples of legal expression-style interpolation:

- `${os}`
- `${$os}`
- `${$os[release]}`
- `${$os['release']}`
- `${$os["release"]}`
- `${$os[2]}` (accesses the value of the third (zero-indexed) key-value pair in the `os` hash)
- `${$osrelease}` (accesses the value of the third key-value pair in the `release` hash)

In the console, an index can be only simple string literals or decimal integer literals. An index cannot include variables or operations (such as string concatenation or integer arithmetic).

Examples of illegal expression-style interpolation:

- `${$:os}`
- `{$os[$release]}`
- `${$os[0xff]}`
- `${$os[6/3]}`
- `${$os[$family + $release]}`
- `${$os + $release}`

Trusted facts

Trusted facts are considered to be keys of a hashmap called `trusted`. This means that all trusted facts must be interpolated using expression-style syntax. For example, the `certname` trusted fact would be expressed like this: `"My name is ${trusted[certname]}"`. Any trusted facts that are themselves structured facts can have further index expressions to access individual fields of that trusted fact.

Note: Regular expressions, resource references, and other keywords (such as `'undef'`) are not supported.

View nodes in a node group

To view all nodes that currently match the rules specified for a node group:

1. In the console, click **Node groups**, and then find the node group that you want to view and select it.
2. Click **Matching nodes**.

You see the number of nodes that match the node group's rules, along with a list of the names of matching nodes. This is based on the facts collected during the node's last Puppet run. The matching nodes list is updated as rules are added, deleted, and edited. Nodes must match rules in ancestor node groups as well as the rules of the current node group in order to be considered a matching node.

Making changes to node groups

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

Edit or remove node groups

You can change the name, description, parent, environment, or environment group setting for node groups, or you can delete node groups that have no children.

1. In the console, click **Node groups**, and select a node group.
2. At the top right of the page, select an option.
 - **Edit node group metadata** — Enables edit mode so you can modify the node group's metadata as needed.
 - **Remove node group** — Removes the node group. You're prompted to confirm the removal.
3. Commit changes.

Remove nodes from a node group

To remove dynamically-assigned nodes from a node group, edit or delete the applicable rule. To remove statically-assigned nodes from a node group, unpin them from the group.

Note: When a node no longer matches the rules of a node group, it is no longer classified with the classes assigned in that node group. However, the resources that were installed by those classes are not removed from the node. For example, if a node group has the `apache` class that installs the Apache package on matching nodes, the Apache package is not removed from the node even when the node no longer matches the node group rules.

1. In the console, click **Node groups**, and select a node group.
2. On the **Rules** tab, select the option appropriate for the type of node.
 - **Dynamically-assigned nodes** — In the **Fact** table, click **Remove** to remove an individual rule, or click **Remove all rules** to delete all rules for the node group.
 - **Statically-assigned nodes** — In the **Certname** table, click **Unpin** to unpin an individual node, or click **Unpin all pinned nodes** to unpin all nodes from the node group.

Tip: To unpin a node from all groups it's pinned to, use the `unpin-from-all` command endpoint.

3. Commit changes.

Related information

[POST /v1/commands/unpin-from-all](#) on page 421

Use the `/v1/commands/unpin-from-all` to unpin specified nodes from all groups they're pinned to. Nodes that are dynamically classified using rules aren't affected.

Remove classes from a node group

Make changes to your node group by removing classes.

Note: If a class appears in a node group list but is crossed out, the class has been deleted from Puppet.

1. In the console, click **Node groups**, and select a node group.
2. On the **Classes** tab, click **Remove this class** to remove an individual class, or click **Remove all classes** to remove all classes from the node group.
3. Commit changes.

Edit or remove parameters

Make changes to your node group by editing or deleting the parameters of a class.

1. In the console, click **Node groups**, and select a node group.
2. On the **Classes** tab, select an option for the class and parameter you want to edit.
 - **Edit** — Enables edit mode so you can modify the parameter as needed.
 - **Remove** — Removes the parameter.
3. Commit changes.

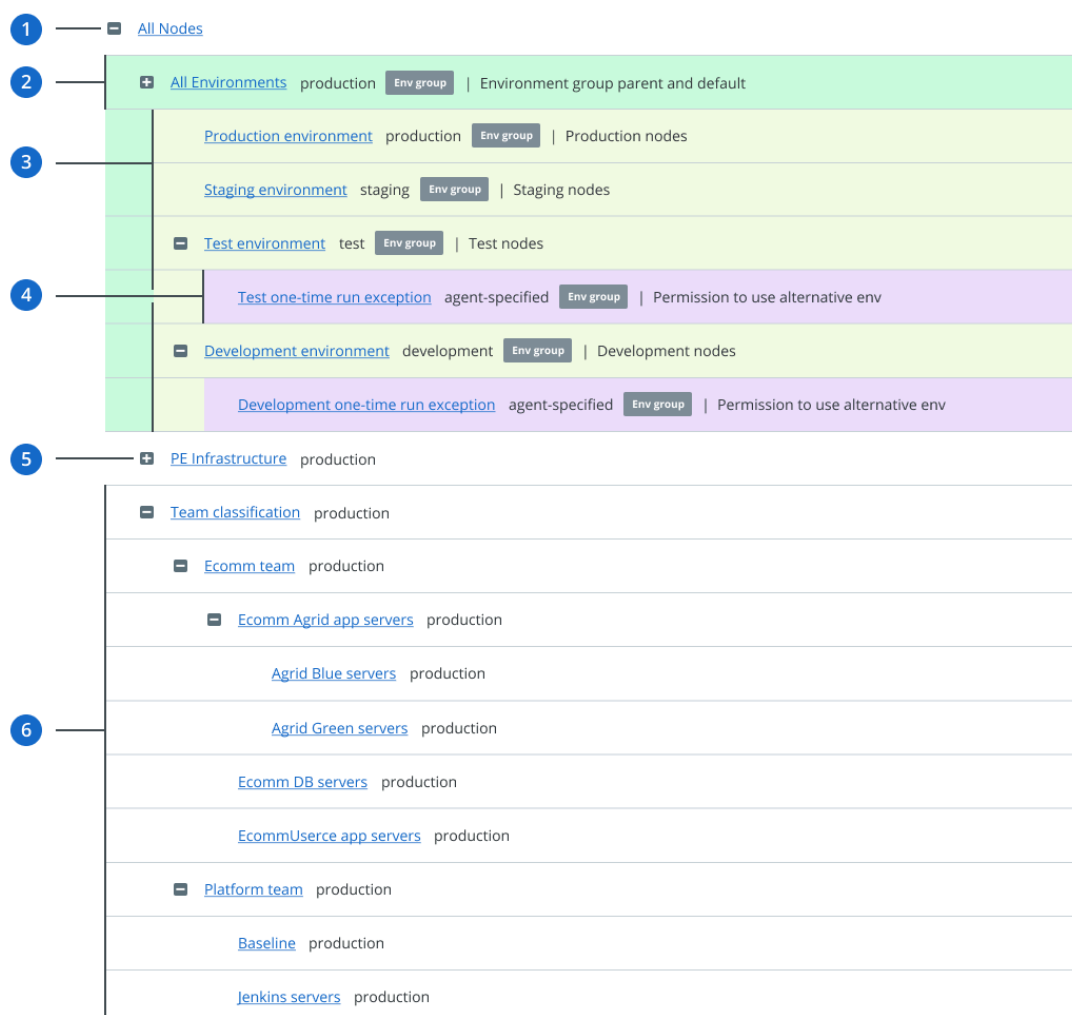
Edit or remove variables

Make changes to your node group by editing or removing variables.

1. In the console, click **Node groups**, and select a node group.
2. On the **Variables** tab, select an option.
 - **Edit** — Enables edit mode so you can modify the variable as needed.
 - **Remove** — Removes the variable.
 - **Remove all variables** — Removes all variables from the node group.
3. Commit changes.

Environment-based testing

An environment-based testing workflow is an effective approach for testing new code before pushing it to production.



1. Root **All Nodes** node group.
2. **All Environments** parent node group. All environment node groups must be children of this group. The Puppet environment assigned to this group is the default Puppet environment used for nodes without an assigned or matched default environment.

Tip: If this group doesn't exist, you should create it.

3. **Environment** node groups. Environment node groups are used to specify the Puppet environment that nodes belong to. Nodes can belong to no more than one environment node group. The group it belongs to determines its default environment.

Each environment node group must meet these conditions:

- Must be a direct child of the **All Environments** node group
- Must not include child groups, except any one-time run exception subgroups used for canary testing
- Must be specified as an environment group in the group metadata
- Must not include classes or configuration data

4. Optional *one-time run exception* group. This group acts as a gatekeeper, and when present, permits nodes in the parent environment group to temporarily use Puppet environments other than their normal default environment.
5. **PE Infrastructure** node group and its children. These are built-in classification node groups used to manage infrastructure nodes. Don't modify these groups except when following official documentation or support instructions.
6. **Classification** node groups. Classification node groups are used to apply classes and configuration data to nodes. Nodes can match more than one classification node group.

Each classification node group must meet these conditions:

- Must be a child of **All Nodes** or another classification group
- Must not be specified as an environment group in the group metadata

Test and promote a parameter

Test and promote a parameter when using an environment-based testing workflow.

1. Create a classification node group with the test environment that is a child of a classification group that uses the production environment.
2. In the child group, set a test parameter. The test parameter overrides the value set by the parent group using the production environment.
3. If you're satisfied with your test results, manually change the parameter in the parent group.
4. Delete the child test group.

Test and promote a class

Test and promote a class when using an environment-based testing workflow.

1. Create a classification node group with the test environment that is a child of a classification group that uses the production environment.
The node classifier validates your parameters against the test environment.
2. Add the class you would like to test.
3. If you're satisfied with your test results, promote your code to production and manually add the class in the parent node group.
4. Delete the child test group.

Test code with canary nodes and alternate Puppet environments

You can test new code using one-time Puppet agent runs on select canary nodes.

Before you begin

You must have configured an environment group other than **All Environments** in which to test code.

To enable the canary workflow, configure a *one-time run exception* child group for each environment group in which you want to test code. This group matches nodes on the fly when you run Puppet with the environment specified.

In this example, we configure a *one-time run exception* environment group as a child of the **Development environment**. You can create a similar group for any environment.

1. Create the *one-time run exception* group with these options:
 - **Name** — `Development one-time run exception`
 - **Parent** — **Development environment**
 - **Environment** — `agent-specified`
 - **Environment group** — `checked`

2. Create a rule to match nodes to this group if they request a Puppet environment other than their default.

- **Fact** — `agent_specified_environment`

Tip: This fact name doesn't autocomplete.

- **Operator** — `~`
- **Value** — `.+`

This rule matches any node from the parent group that requests the use of a non-default environment using either the `--environment` flag on the command line or the `environment` setting in `puppet.conf`.

3. On any node in the **Development environment** group, run `puppet agent -t --environment <ENVNAME>`, where `<ENVNAME>` is the name of the Puppet environment that contains your test code.

If you're using Code Manager and a Git workflow, `<ENVNAME>` is the name of your Git development or feature branch.

During this Puppet run, the agent sets the `agent_specified_environment` value to `<ENVNAME>`. The `Development one-time run exception` group matches the node and permits it to use the requested environment.

Sample rules for one-time run exception groups

These examples show several ways to configure rules for one-time run exception child groups. Where multiple rules are listed, combine the rules by specifying that nodes must match all rules.

Testing scenario	Fact	Operator	Value
Permit any node in the parent environment group to use any Puppet environment	<code>agent_specified_environment</code>	<code>~</code>	<code>.+</code>
Permit RHEL nodes in the parent environment group to use any Puppet environment	<code>agent_specified_environment</code>	<code>~</code>	<code>.+</code>
	<code>facts.os.family</code>	<code>=</code>	<code>RedHat</code>
Permit any nodes in the parent environment group to use any Puppet environment except production	<code>agent_specified_environment</code>	<code>~</code>	<code>.+</code>
	<code>agent_specified_environment</code>	<code>!=</code>	<code>production</code>
Permit any nodes in the parent environment group to use any Puppet environment that begins with the prefix "feature_"	<code>agent_specified_environment</code>	<code>~</code>	<code>^feature_.+</code>

Preconfigured node groups

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

All Nodes node group

This node group is at the top of the hierarchy tree. All other node groups stem from this node group.

Classes

No default classes. Avoid adding classes to this node group.

Matching nodes

All nodes.

Notes

You can't modify the preconfigured rule that matches all nodes.

Infrastructure node groups

Infrastructure node groups are used to manage PE.

Important: Don't make changes to infrastructure node groups other than pinning new nodes for documented functions, like creating compilers. If you want to add custom classifications to infrastructure nodes, create new child groups and apply classification there.

PE Infrastructure node group

This node group is the parent to all other infrastructure node groups.

The **PE Infrastructure** node group contains data such as the hostnames and ports of various services and database info (except for passwords).

It's very important to correctly configure the `puppet_enterprise` class in this node group. The parameters set in this class affect the behavior of all other preconfigured node groups that use classes starting with `puppet_enterprise::profile`. Incorrect configuration of this class could potentially cause a service outage.



CAUTION: Never remove the **PE Infrastructure** node group. Removing the **PE Infrastructure** node group disrupts communication between all of your **PE Infrastructure** nodes.

Classes

`puppet_enterprise` — sets the default parameters for child node groups

Matching nodes

Nodes are not pinned to this node group. The **PE Infrastructure** node group is the parent to other infrastructure node groups, such as **PE Master**, and is used only to set classification that all child node groups inherit. Never pin nodes directly to this node group.

These are the parameters for the `puppet_enterprise` class, where `<YOUR HOST>` is your primary server certname. You can find the certname with `puppet config print certname`.

Parameter	Value
<code>database_host</code>	"<YOUR HOST> "
<code>puppetdb_host</code>	"<YOUR HOST> "
<code>database_port</code>	"<YOUR PORT NUMBER> "
	Required only if you changed the port number from the default 5432.
<code>database_ssl</code>	true if you're using the PE-installed PostgreSQL, and false if you're using your own PostgreSQL.
<code>puppet_master_host</code>	"<YOUR HOST> "
<code>certificate_authority_host</code>	"<YOUR HOST> "
<code>console_port</code>	"<YOUR PORT NUMBER> "
	Required only if you changed the port number from the default 443.)

Parameter	Value
puppetdb_database_name	"pe-puppetdb"
puppetdb_database_user	"pe-puppetdb"
puppetdb_port	"<YOUR PORT NUMBER>" Required only if you changed the port number from the default 8081.)
console_host	"<YOUR HOST>"
pcp_broker_host	"<YOUR HOST>"

PE Certificate Authority node group

This node group is used to manage the certificate authority.

Classes

`puppet_enterprise::profile::certificate_authority` — manages the certificate authority on the primary server

Matching nodes

On a new install, the primary server is pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE Master node group

This node group is used to manage the primary server.

Classes

- `puppet_enterprise::profile::master` — manages the primary server service

Matching nodes

On a new install, the primary server is pinned to this node group.

PE Compiler node group

This node group is a subset of the **PE Master** node group used to manage compilers running the PuppetDB service.

Classes

- `puppet_enterprise::profile::master` — manages the primary server service
- `puppet_enterprise::profile::puppetdb` — manages the PuppetDB service

Matching nodes

Compilers running the PuppetDB service are automatically added to this node group.

Notes

Don't add additional nodes to this node group.

PE Orchestrator node group

This node group is used to manage the application orchestration service.

Classes

`puppet_enterprise::profile::orchestrator` — manages the application orchestration service

Matching nodes

On a new install, the primary server is pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE PuppetDB node group

This node group is used to manage nodes running the PuppetDB service. If the node is also serving as a compiler, it's instead classified in the **PE Compiler** node group.

Classes

`puppet_enterprise::profile::puppetdb` — manages the PuppetDB service

Matching nodes

PuppetDB nodes that aren't functioning as compilers are pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE Console node group

This node group is used to manage the console.

Classes

- `puppet_enterprise::profile::console` — manages the console, node classifier, and RBAC
- `puppet_enterprise::license` — manages the PE license file for the status indicator

Matching nodes

On a new install, the console server node is pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE Agent node group

This node group is used to manage the configuration of agents.

Classes

`puppet_enterprise::profile::agent` — manages your agent configuration

Matching nodes

All managed nodes are pinned to this node group by default.

PE Infrastructure Agent node group

This node group is a subset of the **PE Agent** node group used to manage infrastructure-specific overrides.

Classes

`puppet_enterprise::profile::agent` — manages your agent configuration

Matching nodes

All nodes used to run your Puppet infrastructure and managed by the PE installer are pinned to this node group by default, including the primary server, PuppetDB, console, and compilers.

Notes

You might want to manually pin to this group any additional nodes used to run your infrastructure, such as compiler load balancer nodes. Pinning a compiler load balancer node to this group allows it to receive its catalog from the primary server, rather than the compiler, which helps ensure availability.

PE Database node group

This node group is used to manage the PostgreSQL service.

Classes

- `puppet_enterprise::profile::database` — manages the PE-PostgreSQL service

Matching nodes

The node specified as `puppet_enterprise::database_host` is pinned to this group. By default, the database host is the PuppetDB server node.

Notes

Don't add additional nodes to this node group.

PE Patch Management node group

This is a parent node group for nodes under patch management. Create child node groups based on your needs.

Classes

`pe_patch` — enables patching on nodes.

Matching nodes

There are no nodes pinned to this group. **PE Patch Management** is a parent group for node groups under patch management. You can create node groups with unique configurations based on your patching needs.

Notes

Don't add additional nodes to this node group, only add node groups.

Related information

[Create a node group for nodes under patch management](#) on page 452

Create a node group for nodes you want to patch in PE and add nodes to it. For example, create a node group for testing Windows and *nix patches prior to rolling out patches to other node groups. The **PE Patch Management** parent node group has the `pe_patch` class assigned to it and is in the console by default.

Environment node groups

Environment node groups are used only to set environments. They cannot contain any classification.

Preconfigured environment node groups differ depending on the version of PE you're on and you can customize environment groups as needed for your ecosystem. If you upgrade from an older version of PE, your environment node groups stay the same as they were in the older version.

Managing Windows nodes

You can use PE to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks using Forge modules.

Basic tasks and concepts in Windows

This section is meant to help familiarize you with several common tasks used in Puppet Enterprise (PE) with Windows agents, and explain the concepts and reasons behind performing them.

Practice tasks

In other locations in the documentation, these can be found as steps in tasks, but they are not explained as thoroughly.

Write a simple manifest

Puppet manifest files are lists of resources that have a unique title and a set of named attributes that describe the desired state.

Before you begin

You need a text editor, for example Visual Studio Code (VS Code), to create manifest files. Puppet has an [extension](#) for VS Code that supports syntax highlighting of the Puppet language. Editors like Notepad++ or Notepad won't highlight Puppet syntax, but can also be used to create manifests.

Manifest files are written in Puppet code, a domain specific language (DSL) that defines the desired state of resources on a system, such as files, users, and packages. Puppet compiles these text-based manifests into catalogs, and uses those to apply configuration changes.

1. Create a file named `file.pp` and save it in `c:\myfiles\`
2. With your text editor of choice, add the following text to the file:

```
file { 'c:\\Temp\\foo.txt':
  ensure => present,
  content => 'This is some text in my file'
}
```

Note the following details in this file resource example:

- Puppet uses a basic syntax of `type { title: }`, where `type` is the resource type — in this case it's `file`.
- The resource title (the value before the `:`) is `C:\\Temp\\foo.txt`. The file resource uses the title to determine where to create the file on disk. A resource title must always be unique within a given manifest.
- The `ensure` parameter is set to `present` to create the file on disk, if it's not already present. For `file` type resources, you can also use the value `absent`, which removes the file from disk if it exists.
- The `content` parameter is set to `This is some text in my file`, which writes that value to the file.

Validate your manifest with `puppet parser validate`

You can validate that a manifest's syntax is correct by using the command `puppet parser validate`

1. Check your syntax by entering `puppet parser validate c:\myfiles\file.pp` in the Puppet command prompt. If a manifest has no syntax errors, the tool outputs nothing.
2. To see what output occurs when there is an error, temporarily edit the manifest and remove the `:` after the resource title. Run `puppet parser validate c:\myfiles\file.pp` again, and see the following output:

```
Error: Could not parse for environment production: Syntax error at
'ensure' at c:/myfiles/file.pp:2:3
```

Launch the Puppet command prompt

A lot of common interactions with Puppet are done via the command line.

To open the command line interface, enter `Command Prompt Puppet` in your **Start Menu**, and click **Start Command Prompt with Puppet**.

The Puppet command prompt has a few details worth noting:

- Several important batch files live in the current working directory, `C:\Program Files\Puppet Labs\Puppet\bin`. The most important of these batch files is `puppet.bat`. Puppet is a Ruby based application, and `puppet.bat` is a wrapper around executing Puppet code through `ruby.exe`.
- Running the command prompt with Puppet rather than just the default Windows command prompt ensures that all of the Puppet tooling is in `PATH`, even if you change to a different directory.

Simulate a Puppet run with `--noop`

Puppet has a switch that you can use to test if manifests make the intended changes. This is referred to as non-enforcement or `no-op` mode.

To simulate changes, run `puppet apply c:\myfiles\file.pp --noop` in the command prompt:

```
C:\Program Files\Puppet Labs\Puppet\bin>puppet apply c:\myfiles\file.pp --
noop
Notice: Compiled catalog for win-User.localdomain in environment production
in 0.45 seconds
Notice: /Stage[main]/MainFile[C:\Temp\foo.txt]/ensure: current value absent,
should be present (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
```

```
Notice: Applied catalog in 0.03 seconds
```

Puppet shows you the changes it *would* make, but does not actually make the changes. It *would* create a new file at `C:\Temp\foo.txt`, but it hasn't, because you used `--noop`.

Enforce the desired state with `puppet apply`

When the output of the simulation shows the changes you intend to make, you can start enforcing these changes with the `puppet apply` command.

```
Run puppet apply c:\myfiles\file.pp.
```

To see more details about what this command did, you can specify additional options, such as `--trace`, `--debug`, or `--verbose`, which can help you diagnose problematic code. If `puppet apply` fails, Puppet outputs a full stack trace.

Puppet enforces the resource state you've described in `file.pp`, in this case guaranteeing that a file (`c:\Temp\foo.txt`) is present and has the contents `This is some text in my file`.

Understanding idempotency

A key feature of Puppet is its *idempotency*: the ability to repeatedly apply a manifest to guarantee a desired resource state on a system, with the same results every time.

If a given resource is already in the desired state, Puppet performs no actions. If a given resource is not in the desired state, Puppet takes whatever action is necessary to put the resource into the desired state. Idempotency enables Puppet to simulate resource changes without performing them, and lets you set up configuration management one time, fixing configuration drift without recreating resources from scratch each time Puppet runs.

To demonstrate how Puppet can be applied repeatedly to get the same results, change the manifest at `c:\myfiles\file.pp` to the following:

```
file { 'C:\\Temp\\foo.txt':
  ensure => present,
  content => 'I have changed my file content.'
}
```

Apply the manifest by running `puppet apply c:\myfiles\file.pp`. Open `c:\Temp\foo.txt` and notice that Puppet changes the file's contents.

Applying the manifest again with `puppet apply c:\myfiles\file.pp` results in no changes to the system, demonstrating that Puppet behaves idempotently.

Many of the samples in Puppet documentation assume that you have this basic understanding of creating and editing manifest files, and applying them with `puppet apply`.

Additional command line tools

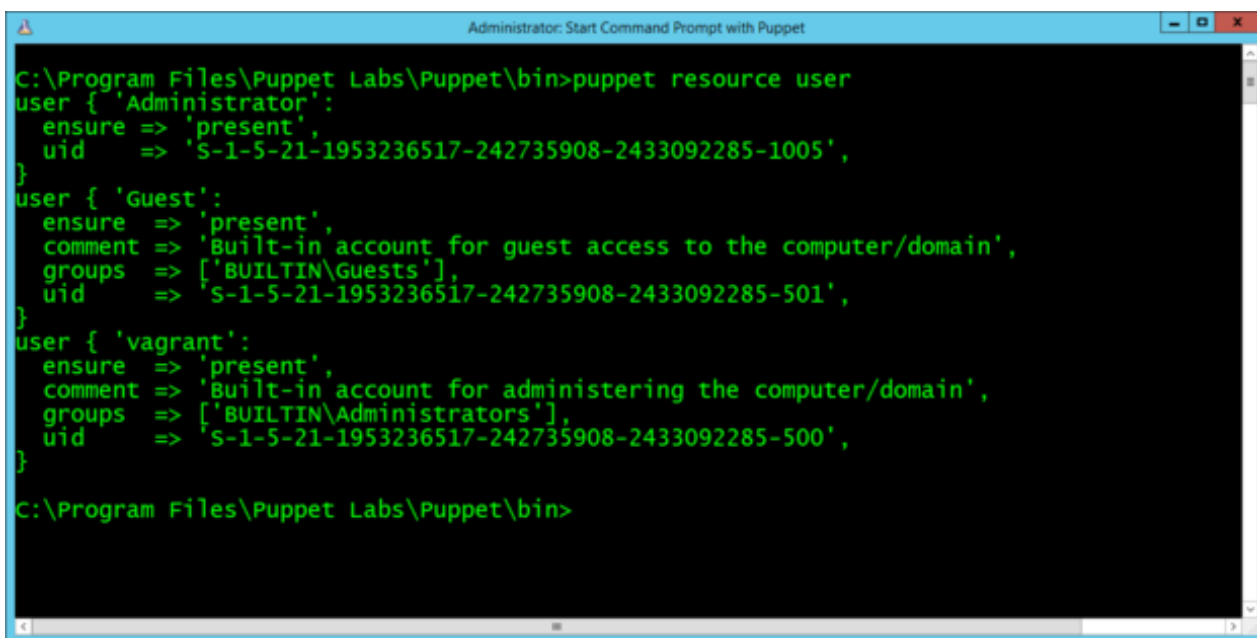
Once you understand how to write manifests, validate them, and use `puppet apply` to enforce your changes, you're ready to use commands such as `puppet agent`, `puppet resource`, and `puppet module install`.

`puppet agent`

Like `puppet apply`, the `puppet agent` command line tool applies configuration changes to a system. However, `puppet agent` retrieves compiled catalogs from a Puppet Server, and applies them to the local system. Puppet is installed as a Windows service, and by default tries to contact the primary server every 30 minutes by running `puppet agent` to retrieve new catalogs and apply them locally.

`puppet resource`

You can run `puppet resource` to query the state of a particular type of resource on the system. For example, to list all of the users on a system, run the command `puppet resource user`.



```

C:\Program Files\Puppet Labs\Puppet\bin>puppet resource user
user { 'Administrator':
  ensure => 'present',
  uid    => 'S-1-5-21-1953236517-242735908-2433092285-1005',
}
user { 'Guest':
  ensure  => 'present',
  comment => 'Built-in account for guest access to the computer/domain',
  groups  => ['BUILTIN\Guests'],
  uid     => 'S-1-5-21-1953236517-242735908-2433092285-501',
}
user { 'vagrant':
  ensure  => 'present',
  comment => 'Built-in account for administering the computer/domain',
  groups  => ['BUILTIN\Administrators'],
  uid     => 'S-1-5-21-1953236517-242735908-2433092285-500',
}
C:\Program Files\Puppet Labs\Puppet\bin>

```

The computer used for this example has three local user accounts: Administrator, Guest, and vagrant. Note that the output is the same format as a manifest, and you can copy and paste it directly into a manifest.

puppet module install

Puppet includes many core resource types, plus you can extend Puppet by installing modules. Modules contain additional resource definitions and the code necessary to modify a system to create, read, modify, or delete those resources. The Puppet Forge contains modules developed by Puppet and community members available for anyone to use.

Puppet synchronizes modules from a primary server to agent nodes during `puppet agent` runs. Alternatively, you can use the standalone Puppet module tool, included when you install Puppet, to manage, view, and test modules.

Run `puppet module list` to show the list of modules installed on the system.

To install modules, the Puppet module tool uses the syntax `puppet module install NAMESPACE/MODULENAME`. The `NAMESPACE` is registered to a module, and `MODULE` refers to the specific module name. A very common module to install on Windows is `registry`, under the `puppetlabs` namespace. So, to install the `registry` module, run `puppet module install puppetlabs/registry`.

Manage Windows services

You can use Puppet to manage Windows services, specifically, to start, stop, enable, disable, list, query, and configure services. This way, you can ensure that certain services are always running or are disabled as necessary.

You write Puppet code to manage services in the manifest. When you apply the manifest, the changes you make to the service are applied.

Note: In addition to using manifests to apply configuration changes, you can query system state using the `puppet resource` command, which emits code as well as applying changes.

Ensure a Windows service is running

There are often services that you always want running in your infrastructure.

To have Puppet ensure that a service is running, use the following code:

```

service { '<service name>':
  ensure => 'running'
}

```

Example

For example, the following manifest code ensures the Windows Time service is running:

```
service { 'w32time':
  ensure => 'running'
}
```

Stop a Windows service

Some services can impair performance, or might need to be stopped for regular maintenance.

To disable a service, use the code:

```
service { '<service name>':
  ensure => 'stopped',
  enable => 'false'
}
```

Example

For example, this disables the disk defragmentation service, which can negatively impact service performance.

```
service { 'defragsvc':
  ensure => 'stopped',
  enable => 'false'
}
```

Schedule a recurring operation with Windows Task Scheduler

Regularly scheduled operations, or tasks, are often necessary on Windows to perform routine system maintenance.

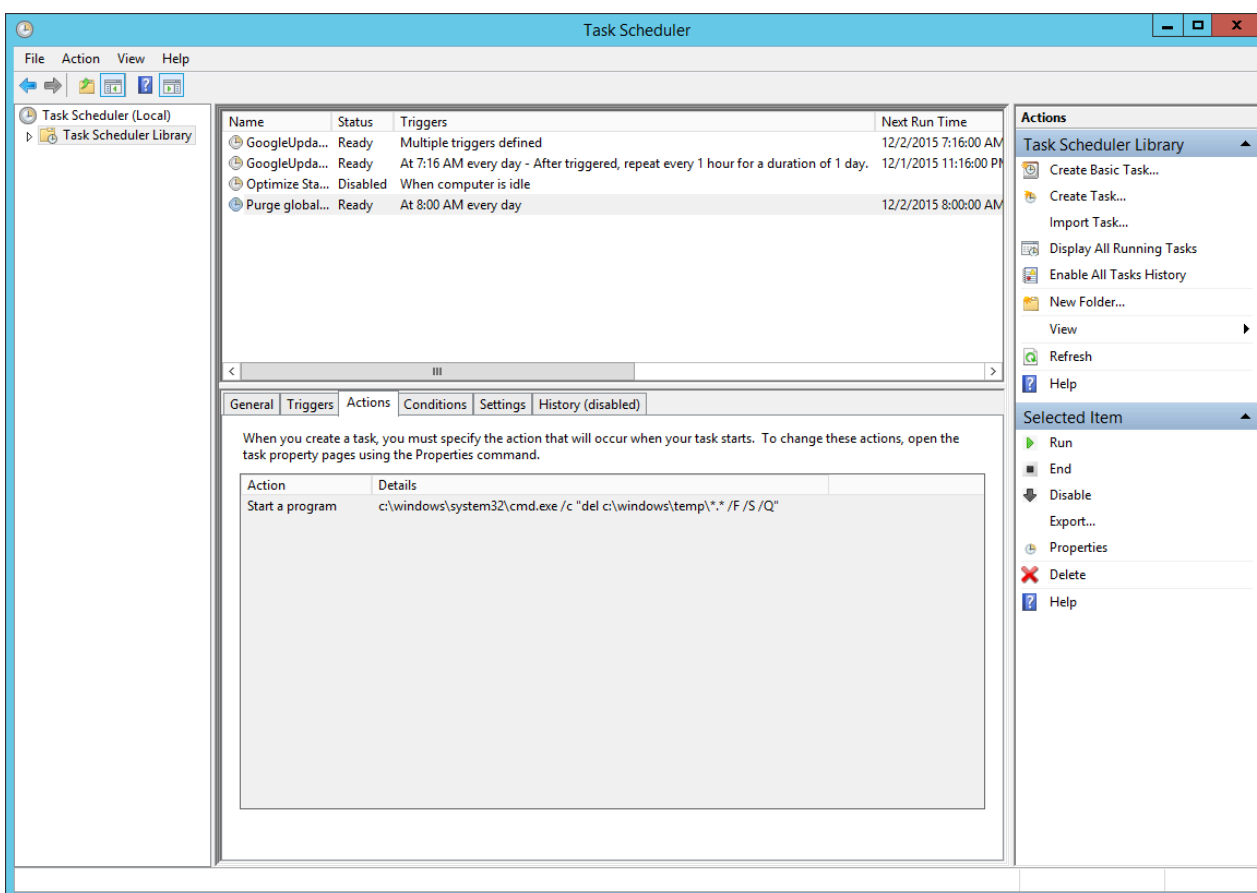
Note: If you need to run an ad-hoc task in the PE console or on the command line, see [Running tasks in PE](#) on page 490.

If you need to sync files from another system on the network, perform backups to another disk, or execute log or index maintenance on SQL Server, you can use Puppet to schedule and perform regular tasks. The following shows how to regularly delete files.

To delete all files recursively from C:\Windows\Temp at 8 AM each day, create a resource called `scheduled_task` with these attributes:

```
scheduled_task { 'Purge global temp files':
  ensure      => present,
  enabled     => true,
  command     => 'c:\windows\system32\cmd.exe',
  arguments   => '/c "del c:\windows\temp\*. * /F /S /Q"',
  trigger     => {
    schedule   => daily,
    start_time => '08:00',
  }
}
```

After you set up Puppet to manage this task, the Task Scheduler includes the task you specified:



Example

In addition to creating a trivial daily task at a specified time, the scheduled task resource supports a number of other more advanced scheduling capabilities, including more fine-tuned scheduling. For example, to change the above task to instead perform a disk clean-up every 2 hours, modify the trigger definition:

```
scheduled_task { 'Purge global temp files every 2 hours':
  ensure => present,
  enabled => true,
  command => 'c:\\windows\\system32\\cmd.exe',
  arguments => '/c "del c:\\windows\\temp\\*.*/F/S/Q"',
  trigger => [{
    day_of_week => ['mon', 'tues', 'wed', 'thurs', 'fri'],
    every => '1',
    minutes_interval => '120',
    minutes_duration => '1440',
    schedule => 'weekly',
    start_time => '07:30'
  }],
  user => 'system',
}
```

You can see the corresponding definition reflected in the Task Scheduler GUI:

Manage Windows users and groups

Puppet can be used to create local group and user accounts. Local user accounts are often desirable for isolating applications requiring unique permissions.

Manage administrator accounts

It is often necessary to standardize the local Windows Administrator password across an entire Windows deployment.

To manage administrator accounts with Puppet, create a user resource with 'Administrator' as the resource title like so:

```
user { 'Administrator':
  ensure => present,
  password => 'yabbadabba'
}
```

Note: Securing the password used in the manifest is beyond the scope of this introductory example, but it's common to use Hiera, a key/value lookup tool for configuration, with eyaml to solve this problem. Not only does this solution provide secure storage for the password value, but it also provides parameterization to support reuse, opening the door to easy password rotation policies across an entire network of Windows machines.

Configure an app to use a different account

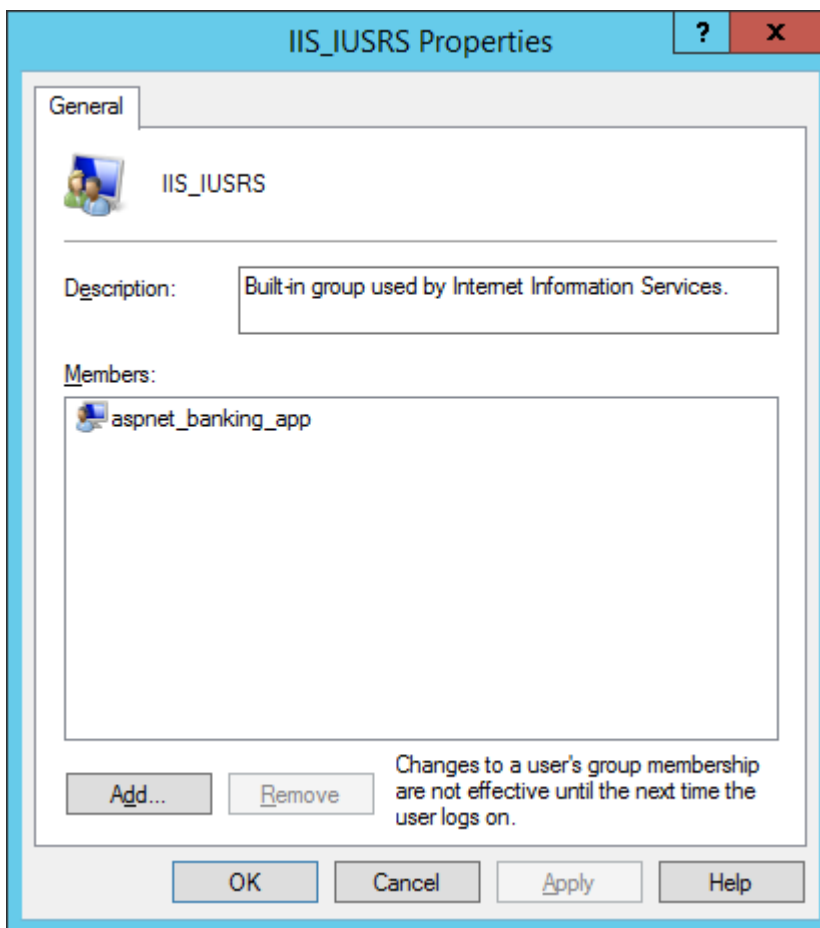
You might not always want to use the default user for an application, you can use Puppet to create users for other applications, like ASP.NET.

To configure ASP.NET apps to use accounts other than the default `Network Service`, create a user and `exec` resource:

```
user { 'aspnet_banking_app':
  ensure      => present,
  managehome  => true,
  comment     => 'ASP.NET Service account for Banking application',
  password    => 'banking_app_password',
  groups      => ['IIS_IUSRS', 'Users'],
  auth_membership => 'minimum',
  notify      => Exec['regiis_aspnet_banking_app']
}

exec { 'regiis_aspnet_banking_app':
  path        => 'c:\\windows\\Microsoft.NET\\Framework\\v4.0.30319',
  command     => 'aspnet_regiis.exe -ga aspnet_banking_app',
  refreshonly => true
}
```

In this example, the user is created in the appropriate groups, and the ASP.NET IIS registration command is run after the user is created to ensure file permissions are correct.



In the user resource, there are a few important details to note:

- `managehome` is set to create the user's home directory on disk.

- `auth_membership` is set to minimum, meaning that Puppet makes sure the `aspnet_banking_app` user is a part of the `IIS_IUSRS` and `Users` group, but doesn't remove the user from any other groups it might be a part of.
- `notify` is set on the user, and `refreshonly` is set on the `exec`. This tells Puppet to run `aspnet_regiis.exe` only when the `aspnet_banking_app` is created or changed.

Manage local groups

Local user accounts are often desirable for isolating applications requiring unique permissions. It can also be useful to manipulate existing local groups.

To add domain users or groups not present in the Domain Administrators group to the local Administrators group, use this code:

```
group { 'Administrators':
  ensure => 'present',
  members => ['DOMAIN\\User'],
  auth_membership => false
}
```

In this case, `auth_membership` is set to false to ensure that `DOMAIN\User` is present in the Administrators group, but that other accounts that might be present in Administrators are not removed.

Note that the `groups` attribute of `user` and the `members` attribute of `group` might both accept SID values, like the well-known SID for Administrators, S-1-5-32-544.

Executing PowerShell code

Some Windows maintenance tasks require the use of Windows Management Instrumentation (WMI), and PowerShell is the most useful way to access WMI methods. Puppet has a special module that can be used to execute arbitrary PowerShell code.

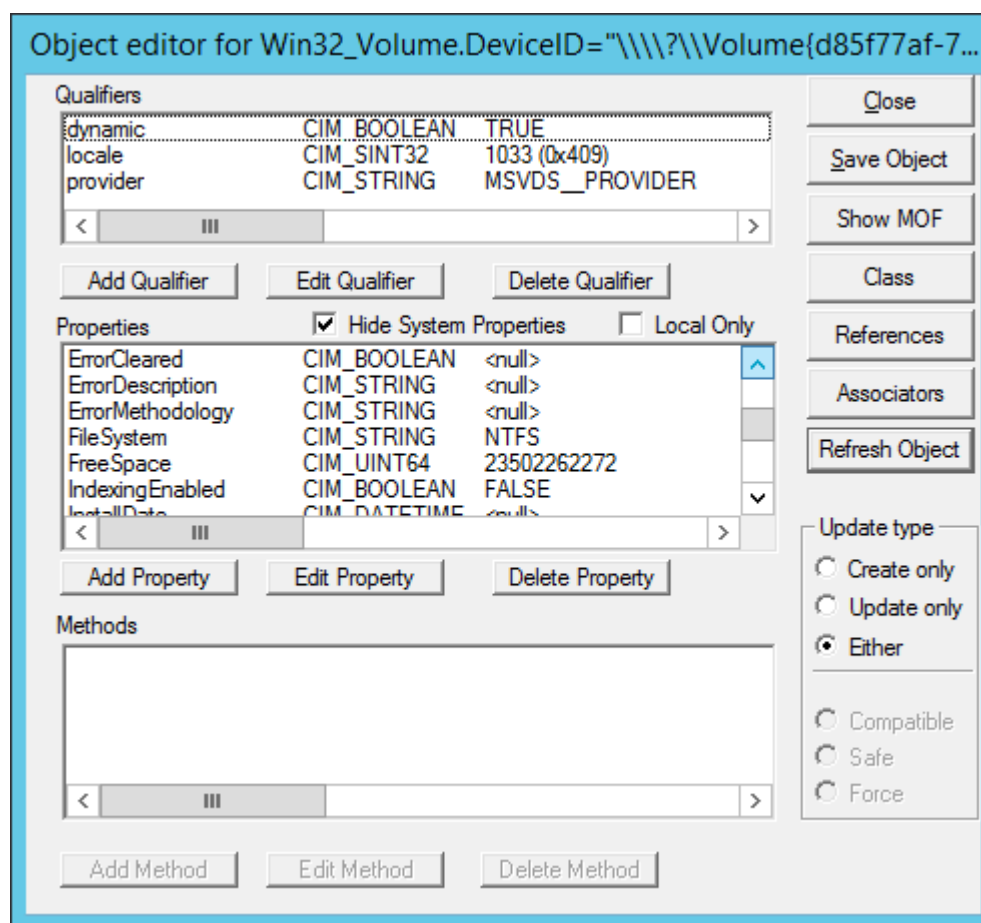
A common Windows maintenance task is to disable Windows drive indexing, because it can negatively impact disk performance on servers.

To disable drive indexing:

```
$drive = 'C:'

exec { 'disable-c-indexing':
  provider => powershell,
  command  => ["\${wmi_volume} = Get-WmiObject -Class Win32_Volume -Filter
'DriveLetter=\"${drive}\"; if (\${wmi_volume.IndexingEnabled -ne \$True)
{ return }; \${wmi_volume} | Set-WmiInstance -Arguments @{IndexingEnabled = \
$False}",
  unless  => "if ((Get-WmiObject -Class Win32_Volume -Filter
'DriveLetter=\"${drive}\"').IndexingEnabled) { exit 1 }",
}
```

You can see the results in your object editor window:



Using the Windows built-in WBEMTest tool, running this manifest sets `IndexingEnabled` to `FALSE`, which is the desired behavior.

This `exec` sets a few important attributes:

- The provider is configured to use PowerShell (which relies on the module).
- The command contains inline PowerShell, and as such, must be escaped with PowerShell variables preceded with `$` must be escaped as `\$`.
- The `unless` attribute is set to ensure that Puppet behaves idempotently, a key aspect of using Puppet to manage resources. If the resource is already in the desired state, Puppet does not modify the resource state.

Using templates to better manage Puppet code

While inline PowerShell is usable as an `exec` resource in your manifest, such code can be difficult to read and maintain, especially when it comes to handling escaping rules.

For executing multi-line scripts, use Puppet templates instead. The following example shows how you can use a template to organize the code for disabling Windows drive indexing.

```
$drive = 'C:'

exec { 'disable-c-indexing':
  command => template('Disable-Indexing.ps1.erb'),
  provider => powershell,
  unless  => "if ((Get-WmiObject -Class Win32_Volume -Filter 'DriveLetter=\\\"$drive\\\"').IndexingEnabled) { exit 1 }",
}
```

The PowerShell code for `Disable-Indexing.ps1.erb` becomes:

```
function Disable-Indexing($Drive)
{
    $drive = Get-WmiObject -Class Win32_Volume -Filter "DriveLetter='$Letter'"
    if ($drive.IndexingEnabled -ne $True) { return }
    $drive | Set-WmiInstance -Arguments @{IndexingEnabled=$False} | Out-Null
}

Disable-Indexing -Drive '<%= @driveLetter %>'
```

Using Windows modules

You can use modules from the Forge to perform basic management tasks on Windows nodes, such as managing access control lists and registry keys, and installing and creating your own packages.

Manage permissions with the `acl` module

The `puppetlabs-acl` module helps you manage access control lists (ACLs), which provide a way to interact with permissions for the Windows file system. This module enables you to set basic permissions up to very advanced permissions using SIDs (Security Identifiers) with an access mask, inheritance, and propagation strategies. First, start with querying some existing permissions.

View file permissions with ACL

ACL is a custom type and provider, so you can use `puppet resource` to look at existing file and folder permissions.

For some types, you can use the command `puppet resource <TYPE NAME>` to get all instances of that type. However, there could be thousands of ACLs on a Windows system, so it's best to specify the folder you want to review the types in. Here, check `c:\Users` to see what permissions it contains.

In the command prompt, enter `puppet resource acl c:\Users`

```
acl { 'c:\Users':
  inherit_parent_permissions => 'false',
  permissions => [
    {identity => 'SYSTEM', rights=> ['full']},
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute'], affects =>
      'self_only'},
    {identity => 'Users', rights => ['read', 'execute'], affects =>
      'children_only'},
    {identity => 'Everyone', rights => ['read', 'execute'], affects =>
      'self_only'},
    {identity => 'Everyone', rights => ['read', 'execute'], affects =>
      'children_only'}
  ],
}
```

As you can see, this particular folder does not inherit permissions from its parent folder; instead, it sets its own permissions and determines how child files and folders inherit the permissions set here.

- `{'identity' => 'SYSTEM', 'rights'=> ['full']}` states that the “SYSTEM” user has full rights to this folder, and by default all children and grandchildren files and folders (as these are the same defaults when creating permissions in Windows).
- `{'identity' => 'Users', 'rights' => ['read', 'execute'], 'affects' => 'self_only'}` gives read and execute permissions to Users but only on the current directory.
- `{'identity' => 'Everyone', 'rights' => ['read', 'execute'], 'affects' => 'children_only'}` gives read and execute permissions to everyone, but only on subfolders and files.

Note: You might see what appears to be the same permission for a user/group twice (both “Users” and “Everyone” above), where one affects only the folder itself and the other is about children only. They are in fact different permissions.

Create a Puppet managed permission

1. Run this code to create your first Puppet managed permission. Then, save it as `perms.pp`

```
file{'c:/tempperms':
  ensure => directory,
}

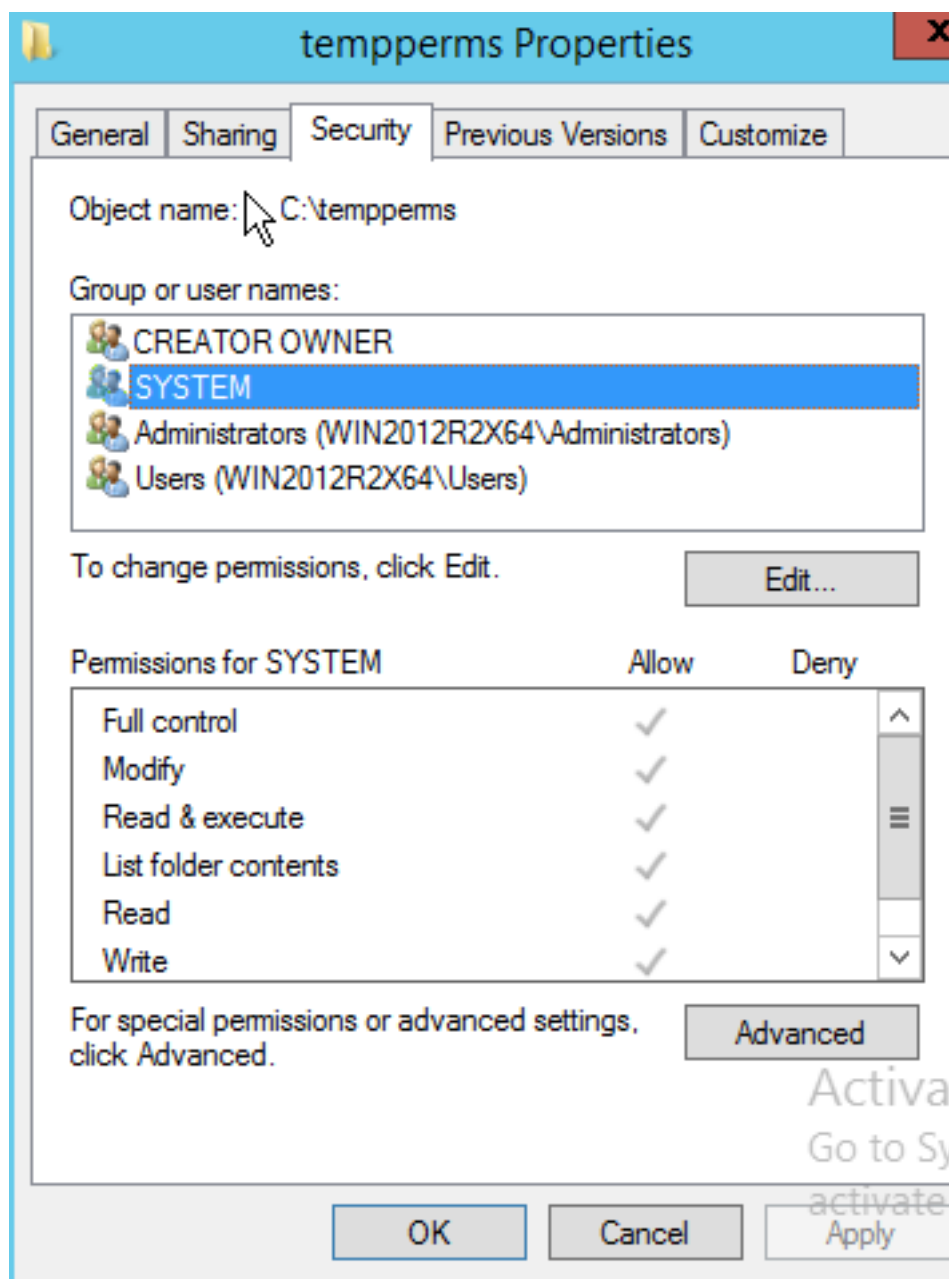
# By default, the acl creates an implicit relationship to any
# file resources it finds that match the location.
acl {'c:/tempperms':
  permissions => [
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read','execute']}
  ],
}
```

2. To validate your manifest, in the command prompt, run `puppet parser validate c:\<FILE PATH>\perms.pp`. If the parser returns nothing, it means validation passed.
3. To apply the manifest, type `puppet apply c:\<FILE PATH>\perms.pp`

Your output should look similar to:

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.12 seconds
Notice: /Stage[main]/Main/File[c:/tempperms]/ensure: created
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions
changed [
] to [
  { identity => 'BUILTIN\Administrators', rights => ["full"] },
  { identity => 'BUILTIN\Users', rights => ["read", "execute"] }
]
Notice: Applied catalog in 0.05 seconds
```

- Review the permissions in your Windows UI. In Windows Explorer, right-click **tempperms** and click **Properties**. Then, click the **Security** tab. It should appear similar to the image below.



- Optional: It might appear that you have more permissions than you were hoping for here. This is because by default Windows inherits parent permissions. In this case, you might not want to do that. Adjust the acl resource to not inherit parent permissions by changing the `perms.pp` file to look like the below by adding `inherit_parent_permissions => false`.

```

acl {'c:/tempperms':
  inherit_parent_permissions => false,
  permissions => [
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute']}
  ],
}
```

6. Save the file, and return the command prompt to run `puppet parser validate c:\<FILE PATH>\perms.pp` again.
7. When it validates, run `puppet apply c:\<FILE PATH>\perms.pp`

You should get output similar to the following:

```
C:\>puppet apply c:\puppet_code\perms.pp
Notice: Compiled catalog for win2012r2x64 in environment production in
0.08 seconds
Notice: /Stage[main]/Main/Acl[c:/tempperms]/inherit_parent_permissions:
inherit_
parent_permissions changed 'true' to 'false'
Notice: Applied catalog in 0.02 seconds
```

8. To check the permissions again, enter `icacls c:\tempperms` in the command prompt. The command, `icacls`, is specifically for displaying and modifying ACLs. The output should be similar to the following:

```
C:\>icacls c:\tempperms
c:\tempperms BUILTIN\Administrators:(OI)(CI)(F)
              BUILTIN\Users:(OI)(CI)(RX)
              NT AUTHORITY\SYSTEM:(OI)(CI)(F)
              BUILTIN\Users:(CI)(AD)
              CREATOR OWNER:(OI)(CI)(IO)(F)
Successfully processed 1 files; Failed processing 0 files
```

The output shows each permission, followed by a list of specific rights in parentheses. This output shows there are more permissions than you specified in `perms.pp`. Puppet manages permissions next to unmanaged or existing permissions. In the case of removing inheritance, by default Windows copies those existing inherited permissions (or Access Control Entries, ACEs) over to the existing ACL so you have some more permissions that you might not want.

9. Remove the extra permissions, so that only the permissions you've specified are on the folder. To do this, in your `perms.pp` set `purge => true` as follows:

```
acl {'c:/tempperms':
  inherit_parent_permissions => false,
  purge                      => true,
  permissions                => [
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute']}
  ],
}
```

10. Run the parser command as you have before. If it still returns no errors, then you can apply the change.

11. To apply the change, run `puppet apply c:\<FILE PATH>\perms.pp`. The output should be similar to below:

```
C:\>puppet apply c:\puppet_code\perms.pp
Notice: Compiled catalog for win2012r2x64 in environment production in
0.08 seco
nds
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions
changed [
{ identity => 'BUILTIN\Administrators', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["read", "execute"] },
{ identity => 'NT AUTHORITY\SYSTEM', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["mask_specific"], mask => '4',
child_types => 'containers' },
{ identity => 'CREATOR OWNER', rights => ["full"], affects =>
'children_only' }
] to [
{ identity => 'BUILTIN\Administrators', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["read", "execute"] }
]
Notice: Applied catalog in 0.05 seconds
```

Puppet outputs a notice as it is removing each of the permissions.

12. Take a look at the output of `icacls` again with `icacls c:\tempperms`

```
c:\>icacls c:\tempperms
c:\tempperms BUILTIN\Administrators:(OI)(CI)(F)
BUILTIN\Users:(OI)(CI)(RX)
Successfully processed 1 files; Failed processing 0 files
```

Now the permissions have been set up for this directory. You can get into more advanced permission scenarios if you read the usage scenarios on this module's Forge page.

Create managed registry keys with `registry` module

You might eventually need to use the registry to access and set highly available settings, among other things. The `puppetlabs-registry` module, which is also a Puppet Supported Module enables you to set both registry keys and values.

View registry keys and values with `puppet resource`

`puppetlabs-registry` is a custom type and provider, so you can use `puppet resource` to look at existing registry settings.

It is also somewhat limited, like the `acl` module in that it is restricted to only what is specified.

1. In your command prompt, run: `puppet resource registry_key 'HKLM\Software\Microsoft\Windows'`

```
C:\>puppet resource registry_key 'HKLM\Software\Microsoft\Windows\'
registry_key { 'HKLM\Software\Microsoft\Windows\':
  ensure => 'present',
}
```

Not that interesting, but now take a look at a registry value.

2. Enter puppet resource registry_value 'HKLM\SYSTEM\CurrentControlSet\Services\BITS\DisplayName'

```
registry_value { 'HKLM\SYSTEM\CurrentControlSet\Services\BITS\DisplayName':
  ensure => 'present',
  data   => ['Background Intelligent Transfer Service'],
  type   => 'string',
}
```

That's a bit more interesting than a registry key.

Keys are like file paths (directories) and values are like files that can have data and be of different types.

Create managed keys

Learn how to make managed registry keys, and see Puppet correct configuration drift when you try and alter them in Registry Editor.

1. Create your first Puppet managed registry keys and values:

```
registry_key { 'HKLM\Software\zTemporaryPuppet':
  ensure => present,
}

# By default the registry creates an implicit relationship to any file
# resources it finds that match the location.
registry_value { 'HKLM\Software\zTemporaryPuppet\StringValue':
  ensure => 'present',
  data   => 'This is a custom value.',
  type   => 'string',
}

#forcing a 32-bit registry view; watch where this is created:
registry_key { '32:HKLM\Software\zTemporaryPuppet':
  ensure => present,
}

registry_value { '32:HKLM\Software\zTemporaryPuppet\StringValue':
  ensure => 'present',
  data   => 'This is a custom 32-bit value.',
  type   => 'expand',
}
```

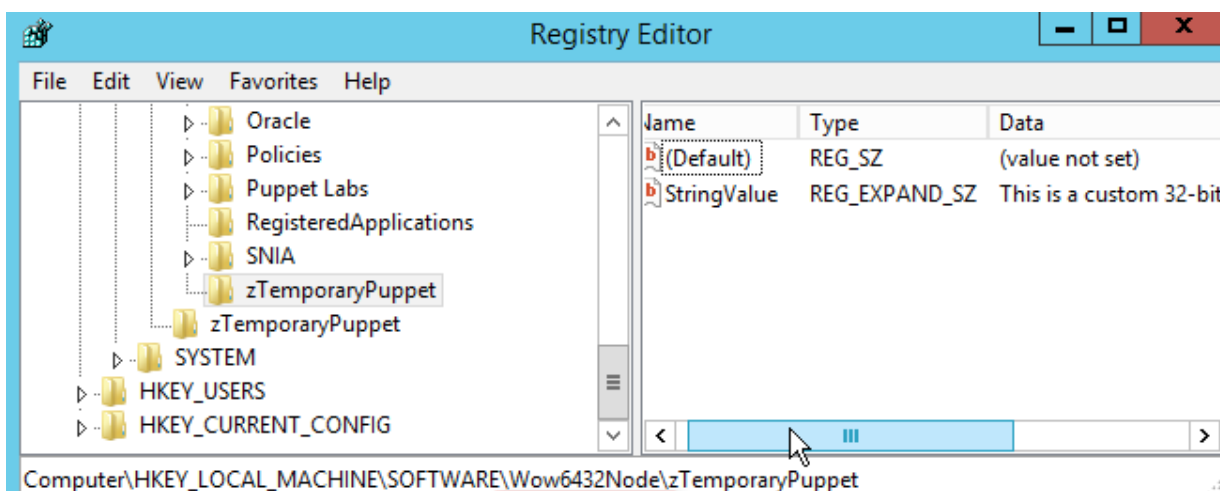
2. Save the file as registry.pp.
3. Validate the manifest. In the command prompt, run puppet parser validate c:\<FILE PATH>\registry.pp

If the parser returns nothing, it means validation passed.

- Now, apply the manifest by running `puppet apply c:\<FILE PATH>\registry.pp` in the command prompt. Your output should look similar to below.

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.11 seconds
Notice: /Stage[main]/Main/Registry_key[HKLM\Software\zTemporaryPuppet]/ensure: created
Notice: /Stage[main]/Main/Registry_value[HKLM\Software\zTemporaryPuppet\StringValue]/ensure: created
Notice: /Stage[main]/Main/Registry_key[32:HKLM\Software\zTemporaryPuppet]/ensure: created
Notice: /Stage[main]/Main/Registry_value[32:HKLM\Software\zTemporaryPuppet\StringValue]/ensure: created
Notice: Applied catalog in 0.03 seconds
```

- Next, inspect the registry and see what you have. Press **Start + R**, then type `regedit` and press **Enter**. Once the **Registry Editor** opens, find your keys under **HKEY_LOCAL_MACHINE**.



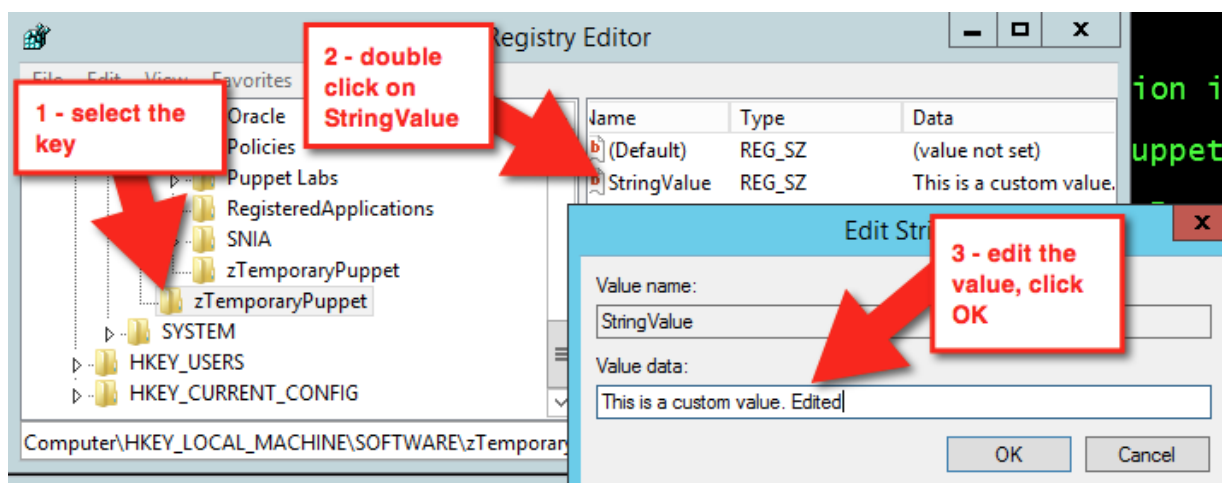
Note that the 32-bit keys were created under the 32-bit section of Wow6432Node for Software.

- Apply the manifest again by running `puppet apply c:\<FILE PATH>\registry.pp`

```
Notice: Compiled catalog for win2012r2x64 in environment production in 0.11 seconds
Notice: Applied catalog in 0.02 seconds
```

Nothing changed, so there is no work for Puppet to do.

7. In **Registry Editor**, change the data. Select **HKLM\Software\zTemporaryPuppet** and in the right box, double-click **StringValue**. Edit the value data, and click **OK**.



This time, changes have been made, so running `puppet apply c:\path\to\registry.pp` results in a different output.

```
Notice: Compiled catalog for win2012r2x64 in environment production
in 0.11 seconds
Notice: /Stage[main]/Main/Registry_value[HKLM\Software\zTemporaryPuppet
\StringValue]/data:
data changed 'This is a custom value. Edited' to 'This is a custom value.'
Notice: Applied catalog in 0.03 seconds
```

Puppet automatically corrects the configuration drift.

8. Next, clean up and remove the keys and values. Make your `registry.pp` file look like the below:

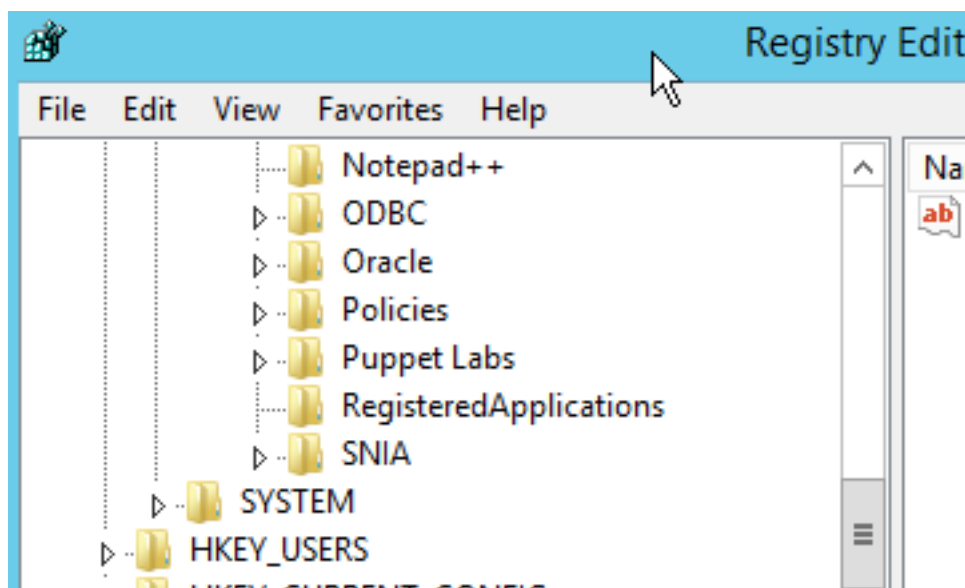
```
registry_key { 'HKLM\Software\zTemporaryPuppet':
  ensure => absent,
}

#forcing a 32 bit registry view, watch where this is created
registry_key { '32:HKLM\Software\zTemporaryPuppet':
  ensure => absent,
}
```

9. Validate it with `puppet parser validate c:\path\to\registry.pp` and apply it again with puppet apply `c:\path\to\registry.pp`

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.06 seconds
Notice: /Stage[main]/Main/Registry_key[HKLM\Software\zTemporaryPuppet]/
ensure: removed
Notice: /Stage[main]/Main/Registry_key[32:HKLM\Software\zTemporaryPuppet]/
ensure
: removed
Notice: Applied catalog in 0.02 seconds
```

Refresh the view in your **Registry Editor**. The values are gone.



Example

Here's a real world example that disables error reporting:

```
class puppetconf::disable_error_reporting {
  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows
Error Reporting\ForceQueue':
    type => dword,
    data => '1',
  }

  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows
Error Reporting\DontShowUI':
    type => dword,
    data => '1',
  }

  registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows
Error Reporting\DontSendAdditionalData':
    type => dword,
    data => '1',
  }

  registry_key { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error
Reporting\Consent':
    ensure => present,
  }
}
```



```

registry_value { 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows
Error Reporting\Consent\DefaultConsent':
  type => dword,
  data => '2',
}
}

```

Create, install, and repackaging packages with the chocolatey module

Chocolatey is a package manager for Windows that is similar in design and execution to package managers on non-Windows systems. The `chocolatey` module is a Puppet Approved Module, so it's not eligible for Puppet Enterprise support services. The module has the capability to install and configure Chocolatey itself, and then manage software on Windows with Chocolatey packages.

View existing packages

Chocolatey has a custom provider for the package resource type, so you can use `puppet resource` to view existing packages.

In the command prompt, run `puppet resource package --param provider | more`

The additional provider parameter in this command outputs all types of installed packages that are detected by multiple providers.

Install Chocolatey

These steps are to install Chocolatey (`choco.exe`) itself. You use the module to ensure Chocolatey is installed.

1. Create a new manifest in the chocolatey module called `chocolatey.pp` with the following contents:

```
include chocolatey
```

2. Validate the manifest by running `puppet parser validate c:\<FILE PATH>\chocolatey.pp` in the command prompt. If the parser returns nothing, it means validation passed.
3. Apply the manifest by running `puppet apply c:\<FILE PATH>\chocolatey.pp`

Your output should look similar to below:

```

Notice: Compiled catalog for win2012r2x64 in environment production in
0.58 seconds
Notice: /Stage[main]/Chocolatey::Install/Windows_env[chocolatey_PATH_env]/
ensure
: created
Notice: /Stage[main]/Chocolatey::Install/
Windows_env[chocolatey_ChocolateyInstal
l_env]/ensure: created
Notice: /Stage[main]/Chocolatey::Install/
Exec[install_chocolatey_official]/retur
ns: executed successfully
Notice: /Stage[main]/Chocolatey::Install/
Exec[install_chocolatey_official]: Trig
gered 'refresh' from 2 events
Notice: Finished catalog run in 13.22 seconds

```

In a production scenario, you're likely to have a `Chocolatey.nupkg` file somewhere internal. In cases like that, you can use the internal `nupkg` for Chocolatey installation:

```

class {'chocolatey':
  chocolatey_download_url => 'https://internalurl/to/chocolatey.nupkg',
  use_7zip                 => false,
  log_output               => true,
}

```

Install a package with chocolatey

Normally, when installing packages you copy them locally first, make any required changes to bring everything they download to an internal location, repackage the package with the edits, and build your own packages to host on your internal package repository (feed). For this exercise, however, you directly install a portable Notepad++ from Chocolatey's community feed. The Notepad++ CommandLine package is portable and shouldn't greatly affect an existing system.

1. Update the manifest `chocolatey.pp` with the following contents:

```
package {'notepadplusplus.commandline':
  ensure => installed,
  provider => chocolatey,
}
```

2. Validate the manifest by running `puppet parser validate c:\<FILE PATH>\chocolatey.pp` in the command prompt. If the parser returns nothing, it means validation passed.
3. Now, apply the manifest with `puppet apply c:\<FILE PATH>\chocolatey.pp`. Your output should look similar to below.

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.75 seconds
Notice: /Stage[main]/Main/Package[notepadplusplus.commandline]/ensure:
created
Notice: Applied catalog in 15.51 seconds
```

If you want to use this package for a production scenario, you need an internal custom feed. This is simple to set up with the `chocolatey_server` module. You could also use Sonatype Nexus, Artifactory, or a CIFS share if you want to host packages with a non-Windows option, or you can use anything on Windows that exposes a NuGet OData feed (Nuget is the packaging infrastructure that Chocolatey uses). See the [How To Host Feed page of the chocolatey wiki](#) for more in-depth information. You could also store packages on your primary server and use a file resource to verify they are in a specific local directory prior to ensuring the packages.

Example

The following example ensures that Chocolatey, the Chocolatey Simple Server (an internal Chocolatey package repository), and some packages are installed. It requires the additional [chocolatey/chocolatey_server module](#).

In `c:\<FILE PATH>\packages` you must have packages for [Chocolatey](#), [Chocolatey.Server](#), [RoundhouseE](#), [Launchy](#), and [Git](#), as well as any of their dependencies for this to work.

```
case $operatingsystem {
  'windows': {
    Package {
      provider => chocolatey,
      source   => 'C:\packages',
    }
  }
}

# include chocolatey
class {'chocolatey':
  chocolatey_download_url => 'file:///C:/packages/
chocolatey.0.9.9.11.nupkg',
  use_7zip                 => false,
  log_output               => true,
}

# This contains the bits to install the custom server.
# include chocolatey_server
class {'chocolatey_server':
  server_package_source => 'C:/packages',
```

```

}

package {'roundhouse':
  ensure => '0.8.5.0',
}

package {'launchy':
  ensure          => installed,
  install_options => ['-override', '-installArgs','', '/VERYSILENT','/
NORESTART',''],
}

package {'git':
  ensure => latest,
}

```

Copy an existing package and make it internal (repackaging packages)

To make the existing package local, use these steps.

Chocolatey's community feed has quite a few packages, but they are geared towards community and use the internet for downloading from official distribution sites. However, they are attractive as they have everything necessary to install a piece of software on your machine. Through the repackaging process, by which you take a community package and bring all of the bits internal or embed them into the package, you can completely internalize a package to host on an internal Chocolatey/NuGet repository. This gives you complete control over a package and removes the aforementioned production trust and control issues.

1. Download the Notepad++ package from Chocolatey's community feed by going to the package page and clicking the download link.
2. Rename the downloaded file to end with `.zip` and unpack the file as a regular archive.
3. Delete the `_rels` and `package` folders and the `[Content_Types].xml` file. These are created during `choco pack` and should not be included, because they're regenerated (and their existence leads to issues).

```

notepadplusplus.commandline.6.8.7.nupkg
####_rels # DELETE
####package # DELETE
# #####services
####tools
### [Content_Types].xml # DELETE
### notepadplusplus.commandline.nuspec

```

4. Open `tools\chocolateyInstall.ps1`.

```

Install-ChocolateyZipPackage 'notepadplusplus.commandline' 'https://
notepad-plus-plus.org/repository/6.x/6.8.7/npp.6.8.7.bin.zip' "$(Split-
Path -parent $MyInvocation.MyCommand.Definition)"

```

5. Download the zip file and place it in the tools folder of the package.
6. Next, edit `chocolateyInstall.ps1` to point to this embedded file instead of reaching out to the internet (if the size of the file is over 50MB, you might want to put it on a file share somewhere internally for better performance).

```

$toolsDir = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
Install-ChocolateyZipPackage 'notepadplusplus.commandline' "$toolsDir
\npp.6.8.7.bin.zip" "$toolsDir"

```

The double quotes allow for string interpolation (meaning variables get interpreted instead of taken literally).

- Next, open the *.nuspec file to view its contents and make any necessary changes.

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/
nuspec.xsd">
  <metadata>
    <id>notepadplusplus.commandline</id>
    <version>6.8.7</version>
    <title>Notepad++ (Portable, CommandLine)</title>
    <authors>Don Ho</authors>
    <owners>Rob Reynolds</owners>
    <projectUrl>https://notepad-plus-plus.org/</projectUrl>
    <iconUrl>https://cdn.rawgit.com/ferrentcoder/chocolatey-
packages/02c21bebe5abb495a56747cbb9b4b5415c933fc0/icons/
notepadplusplus.png</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Notepad++ is a ... </description>
    <summary>Notepad++ is a free (as in "free speech" and also as in "free
beer") source code editor and Notepad replacement that supports several
languages. </summary>
    <tags>notepad notepadplusplus notepad-plus-plus</tags>
  </metadata>
</package>
```

Some organizations change the version field to denote that this is an edited internal package, for example changing 6.8.7 to 6.8.7.20151202. For now, this is not necessary.

- Now you can navigate via the command prompt to the folder with the .nuspec file (from a Windows machine unless you've installed Mono and built choco.exe from source) and use `choco pack`. You can also be more specific and run `choco pack <FILE PATH>\notepadplusplus.commandline.nuspec`. The output should be similar to below.

```
Attempting to build package from 'notepadplusplus.commandline.nuspec'.
Successfully created package 'notepadplusplus.commandline.6.8.7.nupkg'
```

Normally you test on a system to ensure that the package you just built is good prior to pushing the package (just the *.nupkg) to your internal repository. This can be done by using `choco.exe` on a test system to install (`choco install notepadplusplus.commandline -source %cd% -change %cd% to $pwd in PowerShell.exe`) and uninstall (`choco uninstall notepadplusplus.commandline`). Another method of testing is to run the manifest pointed to a local source folder, which is what you are going to do.

- Create `c:\packages` and copy the resulting package file (`notepadplusplus.commandline.6.8.7.nupkg`) into it.

This won't actually install on this system since you just installed the same version from Chocolatey's community feed. So you need to remove the existing package first. To remove it, edit your `chocolatey.pp` to set the package to absent.

```
package {'notepadplusplus.commandline':
  ensure => absent,
  provider => chocolatey,
}
```

- Validate the manifest with `puppet parser validate path\to\chocolatey.pp`. Apply the manifest to ensure the change `puppet apply c:\path\to\chocolatey.pp`.

You can validate that the package has been removed by checking for it in the package install location or by using `choco list -lo`.

11. Update the manifest (`chocolatey.pp`) to use the custom package.

```
package {'notepadplusplus.commandline':
  ensure => latest,
  provider => chocolatey,
  source => 'c:\packages',
}
```

12. Validate the manifest with the parser and then apply it again. You can see Puppet creating the new install in the output.

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.79 seconds
Notice: /Stage[main]/Main/Package[notepadplusplus.commandline]/ensure:
created
Notice: Applied catalog in 14.78 seconds
```

13. In an earlier step, you added a `*.zip` file to the package, so that you can inspect it and be sure the custom package was installed. Navigate to `C:\ProgramData\chocolatey\lib\notepadplusplus.commandline\tools` (if you have a default install location for Chocolatey) and see if you can find the `*.zip` file.

You can also validate the `chocolateyInstall.ps1` by opening and viewing it to see that it is the custom file you changed.

Create a package with chocolatey

Creating your own packages is, for some system administrators, surprisingly simple compared to other packaging standards.

Ensure you have at least Chocolatey CLI (`choco.exe`) version `0.9.9.11` or newer for this next part.

1. From the command prompt, enter `choco new -h` to see a help menu of what the available options are.
2. Next, use `choco new vagrant` to create a package named 'vagrant'. The output should be similar to the following:

```
Creating a new package specification at C:\temppackages\vagrant
Generating template to a file
  at 'C:\temppackages\vagrant\vagrant.nuspec'
Generating template to a file
  at 'C:\temppackages\vagrant\tools\chocolateyinstall.ps1'
Generating template to a file
  at 'C:\temppackages\vagrant\tools\chocolateyuninstall.ps1'
Generating template to a file
  at 'C:\temppackages\vagrant\tools\ReadMe.md'
Successfully generated vagrant package specification files
  at 'C:\temppackages\vagrant'
```

It comes with some files already templated for you to fill out (you can also create your own custom templates for later use).

3. Open `vagrant.nuspec`, and edit it to look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2015/06/
nuspec.xsd">
  <metadata>
    <id>vagrant</id>
    <title>Vagrant (Install)</title>
    <version>1.8.4</version>
    <authors>HashiCorp</authors>
    <owners>my company</owners>
    <description>Vagrant - Development environments made easy.</
description>
  </metadata>
  <files>
    <file src="tools\*" target="tools" />
  </files>
</package>
```

Unless you are sharing with the world, you don't need most of what is in the nuspec template file, so only required items are included above. Match the version of the package in this nuspec file to the version of the underlying software as closely as possible. In this example, Vagrant 1.8.4 is being packaged.

4. Open `chocolateyInstall.ps1` and edit it to look like the following:

```
$ErrorActionPreference = 'Stop';

$packageName= 'vagrant'
$toolsDir    = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
$fileLocation = Join-Path $toolsDir 'vagrant_1.8.4.msi'

$packageArgs = @{
  packageName     = $packageName
  fileType        = 'msi'
  file            = $fileLocation

  silentArgs      = "/qn /norestart"
  validExitCodes= @(0, 3010, 1641)
}

Install-ChocolateyInstallPackage @packageArgs
```

Note: The above is `Install-ChocolateyINSTALLPackage`, not to be confused with `Install-ChocolateyPackage`. The names are very close to each other, however the latter also downloads software from a URI (URL, ftp, file) which is not necessary for this example.

5. Delete the `ReadMe.md` and `chocolateyUninstall.ps1` files. [Download Vagrant](#) and move it to the `tools` folder of the package.

Note: Normally if a package is over 100MB, it is recommended to move the software installer/archive to a share drive and point to it instead. For this example, just bundle it as is.

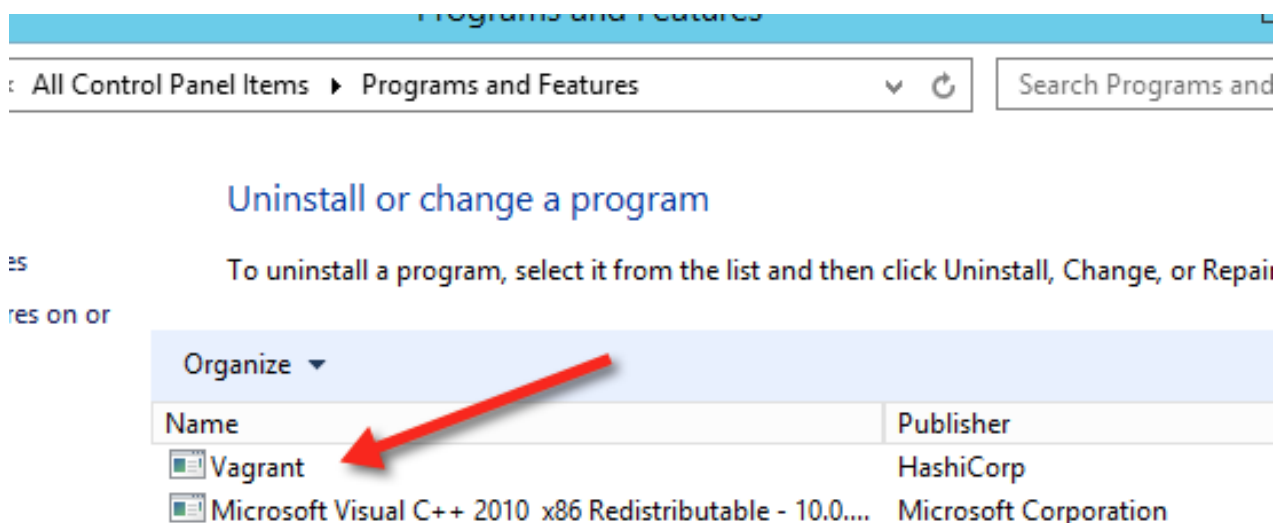
6. Now pack it up by using `choco pack`. Copy the new `vagrant.1.8.4.nupkg` file to `c:\packages`.
7. Open the manifest, and add the new package you just created. Your `chocolatey.pp` file should look like the below.

```
package {'vagrant':
  ensure => installed,
  provider => chocolatey,
  source => 'c:\packages',
}
```

8. Save the file and make sure to validate with the Puppet parser.

9. Use `puppet apply <FILE PATH>\chocolatey.pp` to run the manifest.

10. Open **Control Panel, Programs and Features** and take a look.



Vagrant is installed!

Uninstall packages with Chocolatey

In addition to installing and creating packages, Chocolatey can also help you uninstall them.

To verify that the `choco autoUninstaller` feature is turned on, use `choco feature` to list the features and their current state. If you're using `include chocolatey` or `class chocolatey` to ensure Chocolatey is installed, the configuration is applied automatically (unless you have explicitly disabled it). Starting in Chocolatey version 0.9.10, it is enabled by default.

1. If you see `autoUninstaller - [Disabled]`, you need to enable it. To do this, in the command prompt, run `choco feature enable -n autoUninstaller`. You should see a similar success message:

You should see a similar success message:

```
Enabled autoUninstaller
```

2. To remove Vagrant, edit your `chocolatey.pp` manifest to ensure `=> absent`. Then save and validate the file.

```
package {'vagrant':
  ensure => absent,
  provider => chocolatey,
  source => 'c:\packages',
}
```

3. Next, run `puppet apply <FILE PATH>\chocolatey.pp` to apply the manifest.

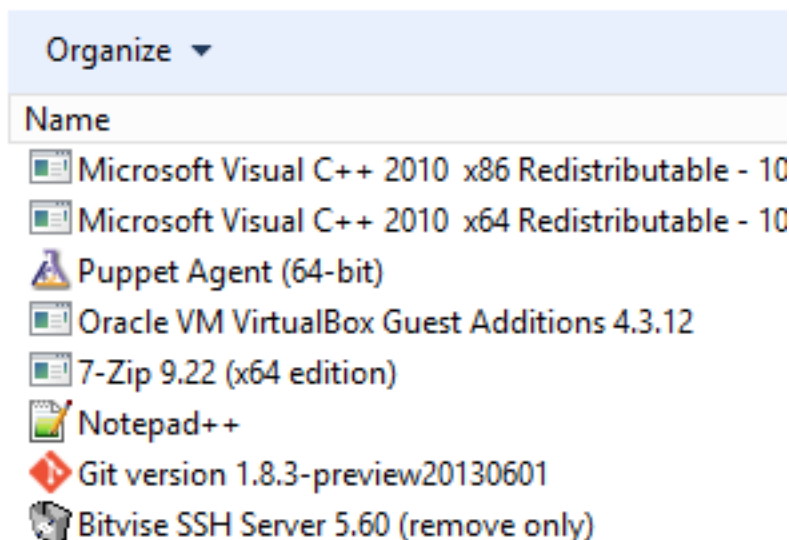
```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.75 seconds
Notice: /Stage[main]/Main/Package[vagrant]/ensure: removed
Notice: Applied catalog in 40.85 seconds
```

You can look in the Control Panel, Programs and Features to see that it's no longer installed!

Uninstall or change a program

To uninstall a program, select it from the list and

1 or



Designing system configs: roles and profiles

Your typical goal with Puppet is to build complete system configurations, which manage all of the software, services, and configuration that you care about on a given system. The roles and profiles method can help keep complexity under control and make your code more reusable, reconfigurable, and refactorable.

- [The roles and profiles method](#) on page 360

The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.

- [Roles and profiles example](#) on page 364

This example demonstrates a complete roles and profiles workflow. Use it to understand the roles and profiles method as a whole. Additional examples show how to design advanced configurations by refactoring this example code to a higher level of complexity.

- [Designing advanced profiles](#) on page 367

In this advanced example, we iteratively refactor our basic roles and profiles example to handle real-world concerns. The final result is — with only minor differences — the Jenkins profile we use in production here at Puppet.

- [Designing convenient roles](#) on page 383

There are several approaches to building roles, and you must decide which ones are most convenient for you and your team.

The roles and profiles method

The *roles and profiles* method is the most reliable way to build reusable, configurable, and refactorable system configurations.

It's not a straightforward recipe: you must think hard about the nature of your infrastructure and your team. It's also not a final state: expect to refine your configurations over time. Instead, it's an approach to *designing your infrastructure's interface* — sealing away incidental complexity, surfacing the significant complexity, and making sure your data behaves predictably.

Building configurations without roles and profiles

Without roles and profiles, people typically build system configurations in their node classifier or main manifest, using Hiera to handle tricky inheritance problems. A standard approach is to create a group of similar nodes and assign classes to it, then create child groups with extra classes for nodes that have additional needs. Another common pattern is to put everything in Hiera, using a very large hierarchy that reflects every variation in the infrastructure.

If this works for you, then it works! You might not need roles and profiles. But most people find direct building gets difficult to understand and maintain over time.

Configuring roles and profiles

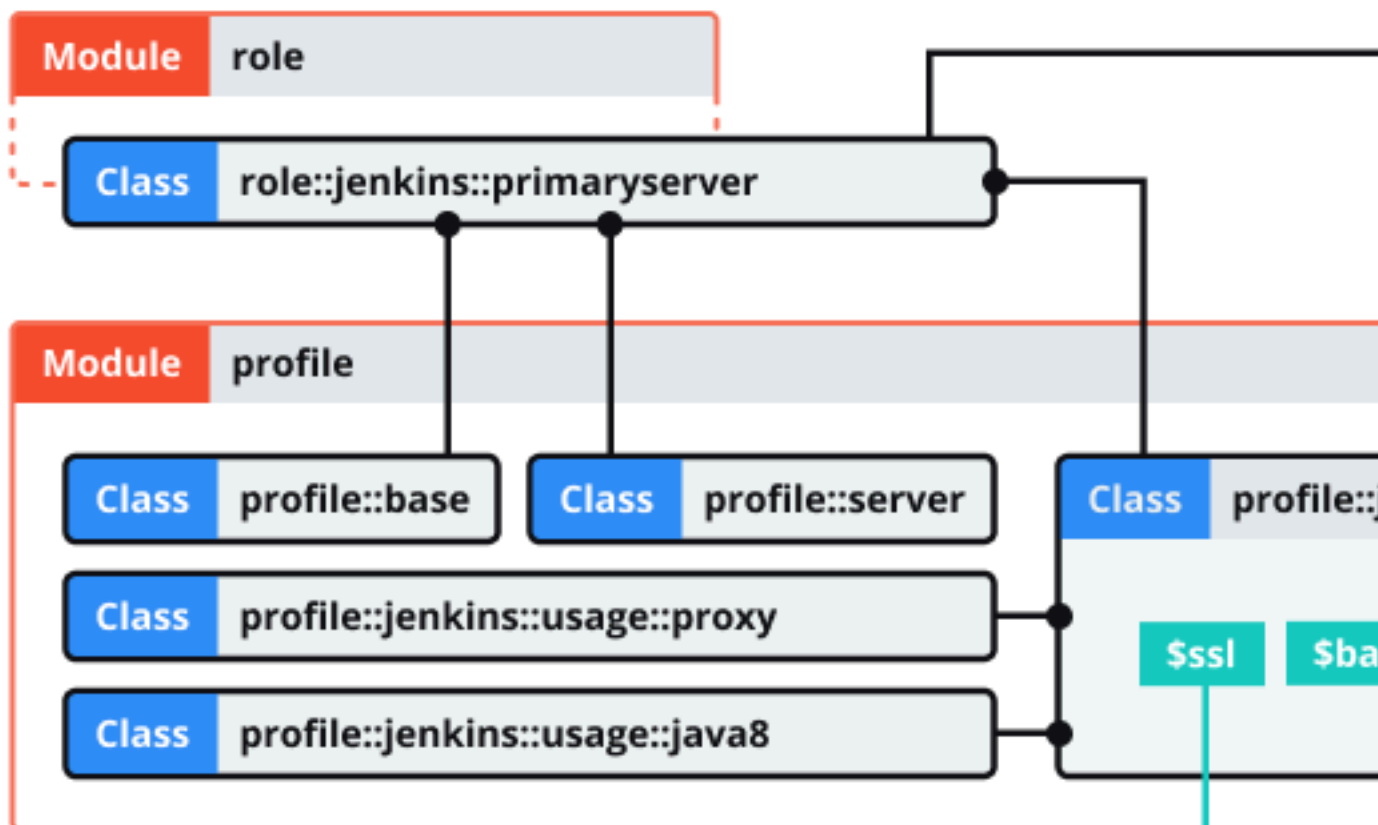
Roles and profiles are *two extra layers of indirection* between your node classifier and your component modules.

The roles and profiles method separates your code into three levels:

- Component modules — Normal modules that manage one particular technology, for example puppetlabs/apache.
- Profiles — Wrapper classes that use multiple component modules to configure a layered technology stack.
- Roles — Wrapper classes that use multiple profiles to build a complete system configuration.

These extra layers of indirection might seem like they add complexity, but they give you a space to build practical, business-specific interfaces to the configuration you care most about. A better interface makes hierarchical data easier to use, makes system configurations easier to read, and makes refactoring easier.

Node Classifier Group



```
Data groups/ci
(...) ::ssl= true
```

```
Data stages/dev
(...) ::backup= false
```

```
Data nodes/ci-primaryserver02-delivery.example.com
(...) ::site_alias= jenkins.example.com
```

In short, from top to bottom:

- Your node classifier assigns one *role* class to a group of nodes. The role manages a whole system configuration, so no other classes are needed. The node classifier does not configure the role in any way.
- That role class declares some *profile* classes with `include`, and does nothing else. For example:

```
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

- Each profile configures a layered technology stack, using multiple component modules and the built-in resource types. (In the diagram, `profile::jenkins::master` uses `puppet/jenkins`, `puppetlabs/apt`, a home-built backup module, and some package and file resources.)
- Profiles can take configuration data from the console, Hiera, or Puppet lookup. (In the diagram, three different hierarchy levels contribute data.)
- Classes from component modules are always declared via a profile, and never assigned directly to a node.
 - If a component class has parameters, you specify them in the profile; never use Hiera or Puppet lookup to override component class params.

Rules for profile classes

There are rules for writing profile classes.

- Make sure you can safely `include` any profile multiple times — don't use resource-like declarations on them.
- Profiles can `include` other profiles.
- Profiles own all the class parameters for their component classes. If the profile omits one, that means you definitely want the default value; the component class shouldn't use a value from Hiera data. If you need to set a class parameter that was omitted previously, refactor the profile.
- There are three ways a profile can get the information it needs to configure component classes:
 - If your business always uses the same value for a given parameter, hardcode it.
 - If you can't hardcode it, try to compute it based on information you already have.
 - Finally, if you can't compute it, look it up in your data. To reduce lookups, identify cases where multiple parameters can be derived from the answer to a single question.

This is a game of trade-offs. Hardcoded parameters are the easiest to read, and also the least flexible. Putting values in your Hiera data is very flexible, but can be very difficult to read: you might have to look through a lot of files (or run a lot of lookup commands) to see what the profile is actually doing. Using conditional logic to derive a value is a middle-ground. Aim for the most readable option you can get away with.

Rules for role classes

There are rules for writing role classes.

- The only thing roles should do is declare profile classes with `include`. Don't declare any component classes or normal resources in a role.

Optionally, roles can use conditional logic to decide which profiles to use.

- Roles should not have any class parameters of their own.
- Roles should not set class parameters for any profiles. (Those are all handled by data lookup.)
- The name of a role should be based on your business's *conversational name* for the type of node it manages.

This means that if you regularly call a machine a "Jenkins master," it makes sense to write a role named `role::jenkins::master`. But if you call it a "web server," you shouldn't use a name like `role::nginx` — go with something like `role::web` instead.

Methods for data lookup

Profiles usually require some amount of configuration, and they must use data lookup to get it.

This profile uses the automatic class parameter lookup to request data.

```
# Example Hiera data
profile::jenkins::jenkins_port: 8000
profile::jenkins::java_dist: jre
profile::jenkins::java_version: '8'

# Example manifest
class profile::jenkins (
  Integer $jenkins_port,
  String  $java_dist,
  String  $java_version
) {
  # ...
}
```

This profile omits the parameters and uses the lookup function:

```
class profile::jenkins {
  $jenkins_port = lookup('profile::jenkins::jenkins_port', {value_type =>
    String, default_value => '9091'})
  $java_dist    = lookup('profile::jenkins::java_dist',      {value_type =>
    String, default_value => 'jdk'})
  $java_version = lookup('profile::jenkins::java_version', {value_type =>
    String, default_value => 'latest'})
  # ...
}
```

In general, class parameters are preferable to lookups. They integrate better with tools like Puppet strings, and they're a reliable and well-known place to look for configuration. But using `lookup` is a fine approach if you aren't comfortable with automatic parameter lookup. Some people prefer the full lookup key to be written in the profile, so they can globally grep for it.

Roles and profiles example

This example demonstrates a complete roles and profiles workflow. Use it to understand the roles and profiles method as a whole. Additional examples show how to design advanced configurations by refactoring this example code to a higher level of complexity.

Configure Jenkins master servers with roles and profiles

Jenkins is a continuous integration (CI) application that runs on the JVM. The Jenkins master server provides a web front-end, and also runs CI tasks at scheduled times or in reaction to events.

In this example, we manage the configuration of Jenkins master servers.

Set up your prerequisites

If you're new to using roles and profiles, do some additional setup before writing any new code.

1. Create two modules: one named `role`, and one named `profile`.

If you deploy your code with Code Manager or r10k, put these two modules in your control repository instead of declaring them in your Puppetfile, because Code Manager and r10k reserve the `modules` directory for their own use.

- a. Make a new directory in the repo named `site`.
- b. Edit the `environment.conf` file to add `site` to the `modulepath`. (For example: `modulepath = site:modules:$basemodulepath`).
- c. Put the `role` and `profile` modules in the `site` directory.

2. Make sure Hiera or Puppet lookup is set up and working, with a hierarchy that works well for you.

Choose component modules

For our example, we want to manage Jenkins itself using the `puppet/jenkins` module.

Jenkins requires Java, and the `puppet/jenkins` module can manage it automatically. But we want finer control over Java, so we're going to disable that. So, we need a Java module, and `puppetlabs/java` is a good choice.

That's enough to start with. We can refactor and expand when we have those working.

To learn more about these modules, see [puppet/jenkins](#) and [puppetlabs/java](#).

Write a profile

From a Puppet perspective, a profile is just a normal class stored in the `profile` module.

Make a new class called `profile::jenkins::master`, located at `.../profile/manifests/jenkins/master.pp`, and fill it with Puppet code.

```
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
  String $jenkins_port = '9091',
  String $java_dist     = 'jdk',
  String $java_version  = 'latest',
) {

  class { ['jenkins']:
    configure_firewall => true,
    install_java       => false,
    port               => $jenkins_port,
    config_hash        => {
      'HTTP_PORT'    => { 'value' => $jenkins_port },
      'JENKINS_PORT' => { 'value' => $jenkins_port },
    },
  },

  class { ['java']:
    distribution => $java_dist,
    version      => $java_version,
    before       => Class['jenkins'],
  }
}
```

This is pretty simple, but is already benefiting us: our interface for configuring Jenkins has gone from 30 or so parameters on the Jenkins class (and many more on the Java class) down to three. Notice that we've hardcoded the `configure_firewall` and `install_java` parameters, and have reused the value of `$jenkins_port` in three places.

Related information

[Rules for profile classes](#) on page 363

There are rules for writing profile classes.

[Methods for data lookup](#) on page 364

Profiles usually require some amount of configuration, and they must use data lookup to get it.

Set data for the profile

Let's assume the following:

- We use some custom facts:
 - `group`: The group this node belongs to. (This is usually either a department of our business, or a large-scale function shared by many nodes.)
 - `stage`: The deployment stage of this node (dev, test, or prod).

- We have a five-layer hierarchy:
 - `console_data` for data defined in the console.
 - `nodes/{trusted.certname}` for per-node overrides.
 - `groups/{facts.group}/{facts.stage}` for setting stage-specific data within a group.
 - `groups/{facts.group}` for setting group-specific data.
 - `common` for global fallback data.
- We have a few one-off Jenkins masters, but most of them belong to the `ci` group.
- Our quality engineering department wants masters in the `ci` group to use the Oracle JDK, but one-off machines can just use the platform's default Java.
- QE also wants their prod masters to listen on port 80.

Set appropriate values in the data, using either Hiera or configuration data in the console.

```
# /etc/puppetlabs/code/environments/production/data/nodes/ci-
master01.example.com.yaml
# --Nothing. We don't need any per-node values right now.

# /etc/puppetlabs/code/environments/production/data/groups/ci/prod.yaml
profile::jenkins::master::jenkins_port: '80'

# /etc/puppetlabs/code/environments/production/data/groups/ci.yaml
profile::jenkins::master::java_dist: 'oracle-jdk8'
profile::jenkins::master::java_version: '8u92'

# /etc/puppetlabs/code/environments/production/data/common.yaml
# --Nothing. Just use the default parameter values.
```

Write a role

To write roles, we consider the machines we'll be managing and decide what else they need in addition to that Jenkins profile.

Our Jenkins masters don't serve any other purpose. But we have some profiles (code not shown) that we expect every machine in our fleet to have:

- `profile::base` must be assigned to every machine, including workstations. It manages basic policies, and uses some conditional logic to include OS-specific profiles as needed.
- `profile::server` must be assigned to every machine that provides a service over the network. It makes sure ops can log into the machine, and configures things like timekeeping, firewalls, logging, and monitoring.

So a role to manage one of our Jenkins masters should include those classes as well.

```
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

Related information

[Rules for role classes](#) on page 363

There are rules for writing role classes.

Assign the role to nodes

Finally, we assign `role::jenkins::master` to every node that acts as a Jenkins master.

Puppet has several ways to assign classes to nodes, so use whichever tool you feel best fits your team. Your main choices are:

- The console node classifier, which lets you group nodes based on their facts and assign classes to those groups.
- The main manifest which can use node statements or conditional logic to assign classes.

- [Hiera](#) or Puppet lookup — Use [the lookup function](#) to do a unique array merge on a special `classes` key, and pass the resulting array to the `include` function.

```
# /etc/puppetlabs/code/environments/production/manifests/site.pp
lookup('classes', {merge => unique}).include
```

Designing advanced profiles

In this advanced example, we iteratively refactor our basic roles and profiles example to handle real-world concerns. The final result is — with only minor differences — the Jenkins profile we use in production here at Puppet.

Along the way, we explain our choices and point out some of the common trade-offs you encounter as you design your own profiles.

Here's the basic Jenkins profile we're starting with:

```
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
  String $jenkins_port = '9091',
  String $java_dist     = 'jdk',
  String $java_version  = 'latest',
) {

  class { ['jenkins']:
    configure_firewall => true,
    install_java       => false,
    port               => $jenkins_port,
    config_hash        => {
      'HTTP_PORT'    => { 'value' => $jenkins_port },
      'JENKINS_PORT' => { 'value' => $jenkins_port },
    },
  },

  class { ['java']:
    distribution => $java_dist,
    version      => $java_version,
    before       => Class['jenkins'],
  }
}
```

Related information

[Rules for profile classes](#) on page 363

There are rules for writing profile classes.

First refactor: Split out Java

We want to manage Jenkins masters *and* Jenkins agent nodes. We won't cover agent profiles in detail, but the first issue we encountered is that they also need Java.

We could copy and paste the Java class declaration; it's small, so keeping multiple copies up-to-date might not be too burdensome. But instead, we decided to break Java out into a separate profile. This way we can manage it one time, then include the Java profile in both the agent and master profiles.

Note: This is a common trade-off. Keeping a chunk of code in only one place (often called the DRY — "don't repeat yourself" — principle) makes it more maintainable and less vulnerable to rot. But it has a cost: your individual profile classes become less readable, and you must view more files to see what a profile actually does. To reduce that readability cost, try to break code out in units that make inherent sense. In this case, the Java profile's job is simple enough to guess by its name — your colleagues don't have to read its code to know that it manages Java 8. Comments can also help.

First, decide how configurable Java needs to be on Jenkins machines. After looking at our past usage, we realized that we use only two options: either we install Oracle's Java 8 distribution, or we default to OpenJDK 7, which the Jenkins module manages. This means we can:

- Make our new Java profile really simple: hardcode Java 8 and take no configuration.
- Replace the two Java parameters from `profile::jenkins::master` with one Boolean parameter (whether to let Jenkins handle Java).

Note: This is rule 4 in action. We reduce our profile's configuration surface by combining multiple questions into one.

Here's the new parameter list:

```
class profile::jenkins::master (
  String $jenkins_port = '9091',
  Boolean $install_jenkins_java = true,
) { # ...
```

And here's how we choose which Java to use:

```
class { 'jenkins':
  configure_firewall => true,
  install_java       => $install_jenkins_java,    # <--- here
  port               => $jenkins_port,
  config_hash        => {
    'HTTP_PORT'      => { 'value' => $jenkins_port },
    'JENKINS_PORT'   => { 'value' => $jenkins_port },
  },
}

# When not using the jenkins module's java version, install java8.
unless $install_jenkins_java { include profile::jenkins::usage::java8 }
```

And our new Java profile:

```
::jenkins::usage::java8
# Sets up java8 for Jenkins on Debian
#
class profile::jenkins::usage::java8 {
  motd::register { 'Java usage profile (profile::jenkins::usage::java8)':' }

  # OpenJDK 7 is already managed by the Jenkins module.
  # ::jenkins::install_java or ::jenkins::agent::install_java should be
  # false to use this profile
  # this can be set through the class parameter $install_jenkins_java
  case $::osfamily {
    'debian': {
      class { 'java':
        distribution => 'oracle-jdk8',
        version      => '8u92',
      }

      package { 'tzdata-java':
        ensure => latest,
      }
    }
  }
  default: {
    notify { "profile::jenkins::usage::java8 cannot set up JDK on
${::osfamily}": }
  }
}
```


Diff of first refactor

```
@@ -1,13 +1,12 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
-   String $jenkins_port = '9091',
-   String $java_dist    = 'jdk',
-   String $java_version = 'latest',
+   String $jenkins_port = '9091',
+   Boolean $install_jenkins_java = true,
) {

    class { 'jenkins':
        configure_firewall => true,
-       install_java       => false,
+       install_java       => $install_jenkins_java,
        port               => $jenkins_port,
        config_hash        => {
            'HTTP_PORT'    => { 'value' => $jenkins_port },
@@ -15,9 +14,6 @@ class profile::jenkins::master (
    },
}

-   class { 'java':
-       distribution => $java_dist,
-       version     => $java_version,
-       before      => Class['jenkins'],
-   }
+   # When not using the jenkins module's java version, install java8.
+   unless $install_jenkins_java { include profile::jenkins::usage::java8 }
}
```

Second refactor: Manage the heap

At Puppet, we manage the Java heap size for the Jenkins app. Production servers didn't have enough memory for heavy use.

The Jenkins module has a `jenkins::sysconfig` defined type for managing system properties, so let's use it:

```
# Manage the heap size on the master, in MB.
if($::memorysize_mb =~ Number and $::memorysize_mb > 8192)
{
    # anything over 8GB we should keep max 4GB for OS and others
    $heap = sprintf('%.0f', $::memorysize_mb - 4096)
} else {
    # This is calculated as 50% of the total memory.
    $heap = sprintf('%.0f', $::memorysize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
    value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
Dhudson.model.DirectoryBrowserSupport.CSP=\\\\"default-src 'self'; img-src
'self'; style-src 'self';\\\"",
}
```

Note: Rule 4 again — we couldn't hardcode this, because we have some smaller Jenkins masters that can't spare the extra memory. But because our production masters are always on more powerful machines, we can calculate the heap based on the machine's memory size, which we can access as a fact. This lets us avoid extra configuration.

Diff of second refactor

```
@@ -16,4 +16,20 @@ class profile::jenkins::master (
    # When not using the jenkins module's java version, install java8.
    unless $install_jenkins_java { include profile::jenkins::usage::java8 }
+
+ # Manage the heap size on the master, in MB.
+ if($::memorysize_mb =~ Number and $::memorysize_mb > 8192)
+ {
+     # anything over 8GB we should keep max 4GB for OS and others
+     $heap = sprintf('%.0f', $::memorysize_mb - 4096)
+ } else {
+     # This is calculated as 50% of the total memory.
+     $heap = sprintf('%.0f', $::memorysize_mb * 0.5)
+ }
+ # Set java params, like heap min and max sizes. See
+ # https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
+ jenkins::sysconfig { 'JAVA_ARGS':
+     value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
+ -XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
+Dhudson.model.DirectoryBrowserSupport.CSP=\\\\"default-src 'self'; img-src
+ 'self'; style-src 'self';\\\\"\"",
+ }
+
+ }
```

Third refactor: Pin the version

We dislike surprise upgrades, so we pin Jenkins to a specific version. We do this with a direct package URL instead of by adding Jenkins to our internal package repositories. Your organization might choose to do it differently.

First, we add a parameter to control upgrades. Now we can set a new value in `.../data/groups/ci/dev.yaml` while leaving `.../data/groups/ci.yaml` alone — our dev machines get the new Jenkins version first, and we can ensure everything works as expected before upgrading our prod machines.

```
class profile::jenkins::master (
    Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/
debian-stable/binary/jenkins_1.642.2_all.deb',
    # ...
) { # ...
```

Then, we set the necessary parameters in the Jenkins class:

```
class { 'jenkins':
    lts                => true,                # <-- here
    repo               => true,                # <-- here
    direct_download    => $direct_download,    # <-- here
    version            => 'latest',            # <-- here
    service_enable     => true,
    service_ensure     => running,
    configure_firewall => true,
    install_java       => $install_jenkins_java,
    port               => $jenkins_port,
    config_hash        => {
        'HTTP_PORT'    => { 'value' => $jenkins_port },
        'JENKINS_PORT' => { 'value' => $jenkins_port },
    },
}
```

This was a good time to explicitly manage the Jenkins *service*, so we did that as well.

Diff of third refactor

```
@@ -1,10 +1,17 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
-   String $jenkins_port = '9091',
-   Boolean $install_jenkins_java = true,
+   String $jenkins_port = '9091',
+   Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-
ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
+   Boolean $install_jenkins_java = true,
) {

    class { 'jenkins':
+     lts                => true,
+     repo               => true,
+     direct_download    => $direct_download,
+     version            => 'latest',
+     service_enable     => true,
+     service_ensure     => running,
+     configure_firewall => true,
+     install_java       => $install_jenkins_java,
+     port               => $jenkins_port,
```

Fourth refactor: Manually manage the user account

We manage a lot of user accounts in our infrastructure, so we handle them in a unified way. The `profile::server` class pulls in `virtual::users`, which has a lot of virtual resources we can selectively realize depending on who needs to log into a given machine.

Note: This has a cost — it's action at a distance, and you need to read more files to see which users are enabled for a given profile. But we decided the benefit was worth it: because all user accounts are written in one or two files, it's easy to see all the users that might exist, and ensure that they're managed consistently.

We're accepting difficulty in one place (where we can comfortably handle it) to banish difficulty in another place (where we worry it would get out of hand). Making this choice required that we know our colleagues and their comfort zones, and that we know the limitations of our existing code base and supporting services.

So, for this example, we change the Jenkins profile to work the same way; we manage the `jenkins` user alongside the rest of our user accounts. While we're doing that, we also manage a few directories that can be problematic depending on how Jenkins is packaged.

Some values we need are used by Jenkins agents as well as masters, so we're going to store them in a `params` class, which is a class that sets shared variables and manages no resources. This is a heavyweight solution, so wait until it provides real value before using it. In our case, we had a lot of OS-specific agent profiles (not shown in these examples), and they made a `params` class worthwhile.

Note: Just as before, "don't repeat yourself" is in tension with "keep it readable." Find the balance that works for you.

```
# We rely on virtual resources that are ultimately declared by
profile::server.
include profile::server

# Some default values that vary by OS:
include profile::jenkins::params
$jenkins_owner      = $profile::jenkins::params::jenkins_owner
$jenkins_group      = $profile::jenkins::params::jenkins_group
$master_config_dir  = $profile::jenkins::params::master_config_dir

file { ['/var/run/jenkins': ensure => 'directory' }
```

```

# Because our account::user class manages the '${master_config_dir}'
directory
# as the 'jenkins' user's homedir (as it should), we need to manage
# `${master_config_dir}/plugins` here to prevent the upstream
# rtyler-jenkins module from trying to manage the homedir as the config
# dir. For more info, see the upstream module's `manifests/plugin.pp`
# manifest.
file { ["${master_config_dir}/plugins":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode   => '0755',
  require => [Group[$jenkins_group], User[$jenkins_owner]],
}

Account::User <| tag == 'jenkins' |>

class { 'jenkins':
  lts           => true,
  repo         => true,
  direct_download => $direct_download,
  version      => 'latest',
  service_enable => true,
  service_ensure => running,
  configure_firewall => true,
  install_java  => $install_jenkins_java,
  manage_user   => false,           # <-- here
  manage_group  => false,           # <-- here
  manage_datadirs => false,         # <-- here
  port          => $jenkins_port,
  config_hash   => {
    'HTTP_PORT' => { 'value' => $jenkins_port },
    'JENKINS_PORT' => { 'value' => $jenkins_port },
  },
}

```

Three things to notice in the code above:

- We manage users with a homegrown `account::user` defined type, which declares a user resource plus a few other things.
- We use an `Account::User` resource collector to realize the Jenkins user. This relies on `profile::server` being declared.
- We set the Jenkins class's `manage_user`, `manage_group`, and `manage_datadirs` parameters to false.
- We're now explicitly managing the `plugins` directory and the `run` directory.

Diff of fourth refactor

```

@@ -5,6 +5,33 @@ class profile::jenkins::master (
    Boolean                                $install_jenkins_java = true,
  ) {

+ # We rely on virtual resources that are ultimately declared by
+ profile::server.
+ include profile::server
+
+ # Some default values that vary by OS:
+ include profile::jenkins::params
+ $jenkins_owner      = $profile::jenkins::params::jenkins_owner
+ $jenkins_group      = $profile::jenkins::params::jenkins_group
+ $master_config_dir  = $profile::jenkins::params::master_config_dir
+
+ file { ['/var/run/jenkins': ensure => 'directory' }

```

```

+
+ # Because our account::user class manages the '${master_config_dir}'
+ directory
+ # as the 'jenkins' user's homedir (as it should), we need to manage
+ # `${master_config_dir}/plugins` here to prevent the upstream
+ # rtyler-jenkins module from trying to manage the homedir as the config
+ # dir. For more info, see the upstream module's `manifests/plugin.pp`
+ # manifest.
+ file { "${master_config_dir}/plugins":
+   ensure => directory,
+   owner  => $jenkins_owner,
+   group  => $jenkins_group,
+   mode   => '0755',
+   require => [Group[$jenkins_group], User[$jenkins_owner]],
+ }
+
+ Account::User <| tag == 'jenkins' |>
+
+ class { 'jenkins':
+   lts           => true,
+   repo         => true,
+@@ -14,6 +41,9 @@ class profile::jenkins::master (
+   service_ensure => running,
+   configure_firewall => true,
+   install_java   => $install_jenkins_java,
+ +   manage_user    => false,
+ +   manage_group   => false,
+ +   manage_datadirs => false,
+   port           => $jenkins_port,
+   config_hash    => {
+     'HTTP_PORT'   => { 'value' => $jenkins_port },

```

Fifth refactor: Manage more dependencies

Jenkins always needs Git installed (because we use Git for source control at Puppet), and it needs SSH keys to access private Git repos and run commands on Jenkins agent nodes. We also have a standard list of Jenkins plugins we use, so we manage those too.

Managing Git is pretty easy:

```

package { 'git':
  ensure => present,
}

```

SSH keys are less easy, because they are sensitive content. We can't check them into version control with the rest of our Puppet code, so we put them in a custom mount point on one specific Puppet server.

Because this server is different from our normal Puppet servers, we made a rule about accessing it: you must look up the hostname from data instead of hardcoding it. This lets us change it in only one place if the secure server ever moves.

```

$secure_server = lookup('puppetlabs::ssl::secure_server')

file { "${master_config_dir}/.ssh":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode   => '0700',
}

file { "${master_config_dir}/.ssh/id_rsa":
  ensure => file,
  owner  => $jenkins_owner,

```

```

    group => $jenkins_group,
    mode  => '0600',
    source => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins",
  }

  file { ["${master_config_dir}/.ssh/id_rsa.pub":
    ensure => file,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0640',
    source => "puppet://${secure_server}/secure/delivery/id_rsa-
jenkins.pub",
  ]
}

```

Plugins are also a bit tricky, because we have a few Jenkins masters where we want to manually configure plugins. So we put the base list in a separate profile, and use a parameter to control whether we use it.

```

class profile::jenkins::master (
  Boolean                                $manage_plugins = false,
  # ...
) {
  # ...
  if $manage_plugins {
    include profile::jenkins::master::plugins
  }
}

```

In the plugins profile, we can use the `jenkins::plugin` resource type provided by the Jenkins module.

```

# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master/plugins.pp
class profile::jenkins::master::plugins {
  jenkins::plugin { 'audit2db': }
  jenkins::plugin { 'credentials': }
  jenkins::plugin { 'jquery': }
  jenkins::plugin { 'job-import-plugin': }
  jenkins::plugin { 'ldap': }
  jenkins::plugin { 'mailer': }
  jenkins::plugin { 'metadata': }
  # ... and so on.
}

```

Diff of fifth refactor

```

@@ -1,6 +1,7 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
  String                                $jenkins_port = '9091',
+ Boolean                                $manage_plugins = false,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-
ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
  Boolean                                $install_jenkins_java = true,
) {
@@ -14,6 +15,20 @@ class profile::jenkins::master (
  $jenkins_group      = $profile::jenkins::params::jenkins_group
  $master_config_dir  = $profile::jenkins::params::master_config_dir

+ if $manage_plugins {
+   # About 40 jenkins::plugin resources:
+   include profile::jenkins::master::plugins
+ }
}

```

```

+
+ # Sensitive info (like SSH keys) isn't checked into version control like
the
+ # rest of our modules; instead, it's served from a custom mount point on
a
+ # designated server.
+ $secure_server = lookup('puppetlabs::ssl::secure_server')
+
+ package { 'git':
+   ensure => present,
+ }
+
+ file { ['/var/run/jenkins': ensure => 'directory' }

+ # Because our account::user class manages the '${master_config_dir}'
directory
@@ -69,4 +84,29 @@ class profile::jenkins::master (
+   value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
Dhudson.model.DirectoryBrowserSupport.CSP=\\\\"default-src 'self'; img-src
'self'; style-src 'self';\\\\"\"",
+ }

+ # Deploy the SSH keys that Jenkins needs to manage its agent machines and
+ # access Git repos.
+ file { ["${master_config_dir}/.ssh":
+   ensure => directory,
+   owner   => $jenkins_owner,
+   group   => $jenkins_group,
+   mode    => '0700',
+ }
+
+ file { ["${master_config_dir}/.ssh/id_rsa":
+   ensure => file,
+   owner   => $jenkins_owner,
+   group   => $jenkins_group,
+   mode    => '0600',
+   source  => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins",
+ }
+
+ file { ["${master_config_dir}/.ssh/id_rsa.pub":
+   ensure => file,
+   owner   => $jenkins_owner,
+   group   => $jenkins_group,
+   mode    => '0640',
+   source  => "puppet://${secure_server}/secure/delivery/id_rsa-
jenkins.pub",
+ }
+
+ }

```

Sixth refactor: Manage logging and backups

Backing up is usually a good idea.

We can use our homegrown backup module, which provides a `backup::job` resource type (profile::server takes care of its prerequisites). But we should make backups optional, so people don't accidentally post junk to our backup server if they're setting up an ephemeral Jenkins instance to test something.

```

class profile::jenkins::master (
  Boolean                                $backups_enabled = false,
  # ...
) {
  # ...

```

```

    if $backups_enabled {
      backup::job { "jenkins-data-${::hostname}":
        files => $master_config_dir,
      }
    }
  }
}

```

Also, our teams gave us some conflicting requests for Jenkins logs:

- Some people want it to use syslog, like most other services.
- Others want a distinct log file so syslog doesn't get spammed, and they want the file to rotate more quickly than it does by default.

That implies a new parameter. We can make one called `$jenkins_logs_to_syslog` and default it to `undef`. If you set it to a standard syslog facility (like `daemon.info`), Jenkins logs there instead of its own file.

We use `jenkins::sysconfig` and our homegrown `logrotate::job` to do the work:

```

class profile::jenkins::master (
  Optional[String[1]] $jenkins_logs_to_syslog = undef,
  # ...
) {
  # ...
  if $jenkins_logs_to_syslog {
    jenkins::sysconfig { 'JENKINS_LOG':
      value => "$jenkins_logs_to_syslog",
    }
  }
  # ...
  logrotate::job { 'jenkins':
    log      => '/var/log/jenkins/jenkins.log',
    options => [
      'daily',
      'copytruncate',
      'missingok',
      'rotate 7',
      'compress',
      'delaycompress',
      'notifempty'
    ],
  }
}

```

Diff of sixth refactor

```

@@ -1,8 +1,10 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
  String $jenkins_port = '9091',
+ Boolean $backups_enabled = false,
  Boolean $manage_plugins = false,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-
ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
+ Optional[String[1]] $jenkins_logs_to_syslog = undef,
  Boolean $install_jenkins_java = true,
) {

@@ -84,6 +86,15 @@ class profile::jenkins::master (
  value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -

```



```

Dhudson.model.DirectoryBrowserSupport.CSP=\\\\"default-src 'self'; img-src
'self'; style-src 'self';\\\\"",
}

+ # Forward jenkins master logs to syslog.
+ # When set to facility.level the jenkins_log uses that value instead of a
+ # separate log file, for example daemon.info
+ if $jenkins_logs_to_syslog {
+   jenkins::sysconfig { 'JENKINS_LOG':
+     value => "$jenkins_logs_to_syslog",
+   }
+ }
+
+ # Deploy the SSH keys that Jenkins needs to manage its agent machines and
+ # access Git repos.
+ file { "${master_config_dir}/.ssh":
@@ -109,4 +120,29 @@ class profile::jenkins::master (
+   source => "puppet://${secure_server}/secure/delivery/id_rsa-
jenkins.pub",
+ }

+ # Back up Jenkins' data.
+ if $backups_enabled {
+   backup::job { "jenkins-data-${::hostname}":
+     files => $master_config_dir,
+   }
+ }

+ # (QENG-1829) Logrotate rules:
+ # Jenkins' default logrotate config retains too much data: by default, it
+ # rotates jenkins.log weekly and retains the last 52 weeks of logs.
+ # Considering we almost never look at the logs, let's rotate them daily
+ # and discard after 7 days to reduce disk usage.
+ logrotate::job { 'jenkins':
+   log      => '/var/log/jenkins/jenkins.log',
+   options => [
+     'daily',
+     'copytruncate',
+     'missingok',
+     'rotate 7',
+     'compress',
+     'delaycompress',
+     'notifempty'
+   ],
+ }
+
+ }

```

Seventh refactor: Use a reverse proxy for HTTPS

We want the Jenkins web interface to use HTTPS, which we can accomplish with an Nginx reverse proxy. We also want to standardize the ports: the Jenkins app always binds to its default port, and the proxy always serves over 443 for HTTPS and 80 for HTTP.

If we want to keep vanilla HTTP available, we can provide an `$ssl` parameter. If set to `false` (the default), you can access Jenkins via both HTTP and HTTPS. We can also add a `$site_alias` parameter, so the proxy can listen on a hostname other than the node's main FQDN.

```

class profile::jenkins::master (
  Boolean          $ssl = false,
  Optional[String[1]] $site_alias = undef,
  # IMPORTANT: notice that $jenkins_port is removed.
  # ...

```

Set `configure_firewall => false` in the Jenkins class:

```
class { 'jenkins':
  lts           => true,
  repo         => true,
  direct_download => $direct_download,
  version      => 'latest',
  service_enable => true,
  service_ensure => running,
  configure_firewall => false,          # <-- here
  install_java  => $install_jenkins_java,
  manage_user   => false,
  manage_group  => false,
  manage_datadirs => false,
  # IMPORTANT: notice that port and config_hash are removed.
}
```

We need to deploy SSL certificates where Nginx can reach them. Because we serve a lot of things over HTTPS, we already had a profile for that:

```
# Deploy the SSL certificate/chain/key for sites on this domain.
include profile::ssl::delivery_wildcard
```

This is also a good time to add some info for the message of the day, handled by puppetlabs/motd:

```
motd::register { 'Jenkins CI master (profile::jenkins::master)': }

if $site_alias {
  motd::register { 'jenkins-site-alias':
    content => @("END"),
              profile::jenkins::master::proxy

              Jenkins site alias: ${site_alias}
              |-END
    order   => 25,
  }
}
```

The bulk of the work is handled by a new profile called `profile::jenkins::master::proxy`. We're omitting the code for brevity; in summary, what it does is:

- Include `profile::nginx`.
- Use resource types from the `jfryman/nginx` to set up a vhost, and to force a redirect to HTTPS if we haven't enabled vanilla HTTP.
- Set up logstash forwarding for access and error logs.
- Include `profile::fw::https` to manage firewall rules, if necessary.

Then, we declare that profile in our main profile:

```
class { 'profile::jenkins::master::proxy':
  site_alias => $site_alias,
  require_ssl => $ssl,
}
```

Important:

We are now breaking rule 1, the most important rule of the roles and profiles method. Why?

Because `profile::jenkins::master::proxy` is a "private" profile that belongs solely to `profile::jenkins::master`. It will never be declared by any role or any other profile.

This is the only exception to rule 1: if you're separating out code *for the sole purpose of readability* --- that is, if you could paste the private profile's contents into the main profile for the exact same effect --- you can use a resource-like declaration on the private profile. This lets you consolidate your data lookups and make the private profile's inputs more visible, while keeping the main profile a little cleaner. If you do this, you must make sure to document that the private profile is private.

If there is any chance that this code might be reused by another profile, obey rule 1.

Diff of seventh refactor

```
@@ -1,8 +1,9 @@
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
class profile::jenkins::master (
- String                      $jenkins_port = '9091',
  Boolean                    $backups_enabled = false,
  Boolean                    $manage_plugins = false,
+ Boolean                    $ssl = false,
+ Optional[String[1]]       $site_alias = undef,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-
ci.org/debian-stable/binary/jenkins_1.642.2_all.deb',
  Optional[String[1]]       $jenkins_logs_to_syslog = undef,
  Boolean                    $install_jenkins_java = true,
@@ -11,6 +12,9 @@ class profile::jenkins::master (
  # We rely on virtual resources that are ultimately declared by
  profile::server.
  include profile::server

+ # Deploy the SSL certificate/chain/key for sites on this domain.
+ include profile::ssl::delivery_wildcard
+
+ # Some default values that vary by OS:
+ include profile::jenkins::params
+ $jenkins_owner = $profile::jenkins::params::jenkins_owner
@@ -22,6 +26,31 @@ class profile::jenkins::master (
  include profile::jenkins::master::plugins
}

+ motd::register { 'Jenkins CI master (profile::jenkins::master)': }
+
+ # This adds the site_alias to the message of the day for convenience when
+ # logging into a server via FQDN. Because of the way motd::register
+ # works, we
+ # need a sort of funny formatting to put it at the end (order => 25) and
+ # to
+ # list a class so there isn't a random "--" at the end of the message.
+ if $site_alias {
+   motd::register { 'jenkins-site-alias':
+     content => @("END"),
+     profile::jenkins::master::proxy
+
+     Jenkins site alias: ${site_alias}
+     |-END
+     order    => 25,
+   }
+ }
+
+ # This is a "private" profile that sets up an Nginx proxy -- it's only
+ # ever
+ # declared in this class, and it would work identically pasted inline.
+ # But because it's long, this class reads more cleanly with it separated
+ # out.
```

```

+ class { 'profile::jenkins::master::proxy':
+   site_alias => $site_alias,
+   require_ssl => $ssl,
+ }
+
# Sensitive info (like SSH keys) isn't checked into version control like
the
# rest of our modules; instead, it's served from a custom mount point on
a
# designated server.
@@ -56,16 +85,11 @@ class profile::jenkins::master (
    version          => 'latest',
    service_enable   => true,
    service_ensure   => running,
-   configure_firewall => true,
+   configure_firewall => false,
    install_java     => $install_jenkins_java,
    manage_user      => false,
    manage_group     => false,
    manage_datadirs  => false,
-   port             => $jenkins_port,
-   config_hash      => {
-     'HTTP_PORT'    => { 'value' => $jenkins_port },
-     'JENKINS_PORT' => { 'value' => $jenkins_port },
-   },
- }

# When not using the jenkins module's java version, install java8.

```

The final profile code

After all of this refactoring (and a few more minor adjustments), here's the final code for `profile::jenkins::master`.

```

# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/master.pp
# Class: profile::jenkins::master
#
# Install a Jenkins master that meets Puppet's internal needs.
#
class profile::jenkins::master (
  Boolean          $backups_enabled = false,
  Boolean          $manage_plugins = false,
  Boolean          $ssl = false,
  Optional[String[1]] $site_alias = undef,
  Variant[String[1], Boolean] $direct_download = 'http://pkg.jenkins-ci.org/
debian-stable/binary/jenkins_1.642.2_all.deb',
  Optional[String[1]] $jenkins_logs_to_syslog = undef,
  Boolean          $install_jenkins_java = true,
) {

  # We rely on virtual resources that are ultimately declared by
profile::server.
  include profile::server

  # Deploy the SSL certificate/chain/key for sites on this domain.
  include profile::ssl::delivery_wildcard

  # Some default values that vary by OS:
  include profile::jenkins::params
  $jenkins_owner      = $profile::jenkins::params::jenkins_owner
  $jenkins_group      = $profile::jenkins::params::jenkins_group
  $master_config_dir  = $profile::jenkins::params::master_config_dir

```

```

if $manage_plugins {
  # About 40 jenkins::plugin resources:
  include profile::jenkins::master::plugins
}

motd::register { 'Jenkins CI master (profile::jenkins::master)':' }

# This adds the site_alias to the message of the day for convenience when
# logging into a server via FQDN. Because of the way motd::register works,
we
# need a sort of funny formatting to put it at the end (order => 25) and
to
# list a class so there isn't a random "---" at the end of the message.
if $site_alias {
  motd::register { 'jenkins-site-alias':
    content => @("END"),
    profile::jenkins::master::proxy

    Jenkins site alias: ${site_alias}
    |-END
    order    => 25,
  }
}

# This is a "private" profile that sets up an Nginx proxy -- it's only
ever
# declared in this class, and it would work identically pasted inline.
# But because it's long, this class reads more cleanly with it separated
out.
class { 'profile::jenkins::master::proxy':
  site_alias => $site_alias,
  require_ssl => $ssl,
}

# Sensitive info (like SSH keys) isn't checked into version control like
the
# rest of our modules; instead, it's served from a custom mount point on a
# designated server.
$secure_server = lookup('puppetlabs::ssl::secure_server')

# Dependencies:
# - Pull in apt if we're on Debian.
# - Pull in the 'git' package, used by Jenkins for Git polling.
# - Manage the 'run' directory (fix for busted Jenkins packaging).
if $::osfamily == 'Debian' { include apt }

package { 'git':
  ensure => present,
}

file { ['/var/run/jenkins': ensure => 'directory' }

# Because our account::user class manages the '${master_config_dir}'
directory
# as the 'jenkins' user's homedir (as it should), we need to manage
# `${master_config_dir}/plugins` here to prevent the upstream
# rtyler-jenkins module from trying to manage the homedir as the config
# dir. For more info, see the upstream module's `manifests/plugin.pp`
# manifest.
file { "${master_config_dir}/plugins":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
}

```

```

    mode      => '0755',
    require => [Group[$jenkins_group], User[$jenkins_owner]],
  }

Account::User <| tag == 'jenkins' |>

class { 'jenkins':
  lts           => true,
  repo          => true,
  direct_download => $direct_download,
  version       => 'latest',
  service_enable => true,
  service_ensure => running,
  configure_firewall => false,
  install_java  => $install_jenkins_java,
  manage_user   => false,
  manage_group  => false,
  manage_datadirs => false,
}

# When not using the jenkins module's java version, install java8.
unless $install_jenkins_java { include profile::jenkins::usage::java8 }

# Manage the heap size on the master, in MB.
if($::memorysize_mb =~ Number and $::memorysize_mb > 8192)
{
  # anything over 8GB we should keep max 4GB for OS and others
  $heap = sprintf('%.0f', $::memorysize_mb - 4096)
} else {
  # This is calculated as 50% of the total memory.
  $heap = sprintf('%.0f', $::memorysize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
  value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
Dhudson.model.DirectoryBrowserSupport.CSP=\\\\"default-src 'self'; img-src
'self'; style-src 'self';\\\\"\"",
}

# Forward jenkins master logs to syslog.
# When set to facility.level the jenkins_log uses that value instead of a
# separate log file, for example daemon.info
if $jenkins_logs_to_syslog {
  jenkins::sysconfig { 'JENKINS_LOG':
    value => "$jenkins_logs_to_syslog",
  }
}

# Deploy the SSH keys that Jenkins needs to manage its agent machines and
# access Git repos.
file { "${master_config_dir}/.ssh":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode   => '0700',
}

file { "${master_config_dir}/.ssh/id_rsa":
  ensure => file,
  owner  => $jenkins_owner,
  group  => $jenkins_group,

```

```

    mode    => '0600',
    source => "puppet://${secure_server}/secure/delivery/id_rsa-jenkins",
  }

  file { "${master_config_dir}/.ssh/id_rsa.pub":
    ensure => file,
    owner  => $jenkins_owner,
    group  => $jenkins_group,
    mode   => '0640',
    source => "puppet://${secure_server}/secure/delivery/id_rsa-
jenkins.pub",
  }

  # Back up Jenkins' data.
  if $backups_enabled {
    backup::job { "jenkins-data-${::hostname}":
      files => $master_config_dir,
    }
  }

  # (QENG-1829) Logrotate rules:
  # Jenkins' default logrotate config retains too much data: by default, it
  # rotates jenkins.log weekly and retains the last 52 weeks of logs.
  # Considering we almost never look at the logs, let's rotate them daily
  # and discard after 7 days to reduce disk usage.
  logrotate::job { 'jenkins':
    log      => '/var/log/jenkins/jenkins.log',
    options => [
      'daily',
      'copytruncate',
      'missingok',
      'rotate 7',
      'compress',
      'delaycompress',
      'notifempty'
    ],
  }
}

```

Designing convenient roles

There are several approaches to building roles, and you must decide which ones are most convenient for you and your team.

High-quality roles strike a balance between readability and maintainability. For most people, the benefit of seeing the entire role in a single file outweighs the maintenance cost of repetition. Later, if you find the repetition burdensome, you can change your approach to reduce it. This might involve combining several similar roles into a more complex role, creating sub-roles that other roles can include, or pushing more complexity into your profiles.

So, begin with granular roles and deviate from them only in small, carefully considered steps.

Here's the basic Jenkins role we're starting with:

```

class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}

```

Related information

[Rules for role classes](#) on page 363

There are rules for writing role classes.

First approach: Granular roles

The simplest approach is to make one role per type of node, period. For example, the Puppet Release Engineering (RE) team manages some additional resources on their Jenkins masters.

With granular roles, we'd have at least two Jenkins master roles. A basic one:

```
class role::jenkins::master {
  include profile::base
  include profile::server
  include profile::jenkins::master
}
```

...and an RE-specific one:

```
class role::jenkins::master::release {
  include profile::base
  include profile::server
  include profile::jenkins::master
  include profile::jenkins::master::release
}
```

The benefits of this setup are:

- Readability — By looking at a single class, you can immediately see which profiles make up each type of node.
- Simplicity — Each role is just a linear list of profiles.

Some drawbacks are:

- Role bloat — If you have a lot of only-slightly-different nodes, you quickly have a large number of roles.
- Repetition — The two roles above are almost identical, with one difference. If they're two separate roles, it's harder to see how they're related to each other, and updating them can be more annoying.

Second approach: Conditional logic

Alternatively, you can use conditional logic to handle differences between closely-related kinds of nodes.

```
class role::jenkins::master::release {
  include profile::base
  include profile::server
  include profile::jenkins::master

  if $facts['group'] == 'release' {
    include profile::jenkins::master::release
  }
}
```

The benefits of this approach are:

- You have fewer roles, and they're easy to maintain.

The drawbacks are:

- Reduced readability...maybe. Conditional logic isn't usually hard to read, especially in a simple case like this, but you might feel tempted to add a bunch of new custom facts to accommodate complex roles. This can make roles much harder to read, because a reader must also know what those facts mean.

In short, be careful of turning your node classification system inside-out. You might have a better time if you separate the roles and assign them with your node classifier.

Third approach: Nested roles

Another way of reducing repetition is to let roles include other roles.

```
class role::jenkins::master {
  # Parent role:
  include role::server
  # Unique classes:
  include profile::jenkins::master
}

class role::jenkins::master::release {
  # Parent role:
  include role::jenkins::master
  # Unique classes:
  include profile::jenkins::master::release
}
```

In this example, we reduce boilerplate by having `role::jenkins::master` include `role::server`. When `role::jenkins::master::release` includes `role::jenkins::master`, it automatically gets `role::server` as well. With this approach, any given role only needs to:

- Include the "parent" role that it most resembles.
- Include the small handful of classes that differentiate it from its parent.

The benefits of this approach are:

- You have fewer roles, and they're easy to maintain.
- Increased visibility in your node classifier.

The drawbacks are:

- Reduced readability: You have to open more files to see the real content of a role. This isn't much of a problem if you go only one level deep, but it can get cumbersome around three or four.

Fourth approach: Multiple roles per node

In general, we recommend that you assign only one role to a node. In an infrastructure where nodes usually provide one primary service, that's the best way to work.

However, if your nodes tend to provide more than one primary service, it can make sense to assign multiple roles.

For example, say you have a large application that is usually composed of an application server, a database server, and a web server. To enable lighter-weight testing during development, you've decided to provide an "all-in-one" node type to your developers. You could do this by creating a new `role::our_application::monolithic` class, which includes all of the profiles that compose the three normal roles, but you might find it simpler to use your node classifier to assign all three roles (`role::our_application::app`, `role::our_application::db`, and `role::our_application::web`) to those all-in-one machines.

The benefit of this approach are:

- You have fewer roles, and they're easy to maintain.

The drawbacks are:

- There's no actual "role" that describes your multi-purpose nodes; instead, the source of truth for what's on them is spread out between your roles and your node classifier, and you must cross-reference to understand their configurations. This reduces readability.
- The normal and all-in-one versions of a complex application are likely to have other subtle differences you need to account for, which might mean making your "normal" roles more complex. It's possible that making a separate role for this kind of node would *reduce* your overall complexity, even though it increases the number of roles and adds repetition.

Fifth approach: Super profiles

Because profiles can already include other profiles, you can decide to enforce an additional rule at your business: all profiles must include any other profiles needed to manage a complete node that provides that service.

For example, our `profile::jenkins::master` class could include both `profile::server` and `profile::base`, and you could manage a Jenkins master server by directly assigning `profile::jenkins::master` in your node classifier. In other words, a "main" profile would do all the work that a role usually does, and the roles layer would no longer be necessary.

The benefits of this approach are:

- The chain of dependencies for a complex service can be more clear this way.
- Depending on how you conceptualize code, this can be easier in a lot of ways!

The drawbacks are:

- Loss of flexibility. This reduces the number of ways in which your roles can be combined, and reduces your ability to use alternate implementations of dependencies for nodes with different requirements.
- Reduced readability, on a much grander scale. Like with nested roles, you lose the advantage of a clean, straightforward list of what a node consists of. Unlike nested roles, you also lose the clear division between "top-level" complete system configurations (roles) and "mid-level" groupings of technologies (profiles). Not every profile makes sense as an entire system, so you some way to keep track of which profiles are the top-level ones.

Some people really find continuous hierarchies easier to reason about than sharply divided layers. If everyone in your organization is on the same page about this, a "profiles and profiles" approach might make sense. But we strongly caution you against it unless you're very sure; for most people, a true roles and profiles approach works better. Try the well-traveled path first.

Sixth approach: Building roles in the node classifier

Instead of building roles with the Puppet language and then assigning them to nodes with your node classifier, you might find your classifier flexible enough to build roles directly.

For example, you might create a "Jenkins masters" group in the console and assign it the `profile::base`, `profile::server`, and `profile::jenkins::master` classes, doing much the same job as our basic `role::jenkins::master` class.

Important:

If you're doing this, make sure you don't set parameters for profiles in the classifier. Continue to use Hiera / Puppet lookup to configure profiles.

This is because profiles are allowed to include other profiles, which interacts badly with the resource-like behavior that node classifiers use to set class parameters.

The benefits of this approach are:

- Your node classifier becomes much more powerful, and can be a central point of collaboration for managing nodes.
- Increased readability: A node's page in the console displays the full content of its role, without having to cross-reference with manifests in your `role` module.

The drawbacks are:

- Loss of flexibility. The Puppet language's conditional logic is often more flexible and convenient than most node classifiers, including the console.
- Your roles are no longer in the same code repository as your profiles, and it's more difficult to make them follow the same code promotion processes.

Node classifier API v1

These are the endpoints for the node classifier v1 API.

Tip: In addition to these endpoints, you can use the status API to check the health of the node classifier service.

- [Forming node classifier requests](#) on page 388

Requests to the node classifier API must be well-formed HTTP(S) requests.

- [Groups endpoint](#) on page 389

The `groups` endpoint is used to create, read, update, and delete groups.

- [Groups endpoint examples](#) on page 407

Use example requests to better understand how to work with groups in the node classifier API.

- [Classes endpoint](#) on page 409

Use the `classes` endpoints to retrieve lists of classes, including classes with specific environments. The output from this endpoint is especially useful for creating new node groups, which usually contain a reference to one or more classes.

- [Classification endpoint](#) on page 411

The `classification` endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

- [Commands endpoint](#) on page 421

Use the `commands` endpoint to unpin specified nodes from all groups they're pinned to.

- [Environments endpoint](#) on page 422

Use the `environments` endpoint to retrieve information about environments in the node classifier. The output tells you either which environments are available or whether a named environment exists. The output can be helpful when creating new node groups, which must be associated with an environment. The node classifier gets its information about environments from Puppet, so do not use this endpoint to create, update, or delete them.

- [Nodes check-in history endpoint](#) on page 423

Use the `nodes` endpoint to retrieve historical information about nodes that have checked into the node classifier.

- [Group children endpoint](#) on page 426

Use the `group children` endpoint to retrieve a specified group and its descendents.

- [Rules endpoint](#) on page 429

Use the `rules` endpoint to translate a group's rule condition into PuppetDB query syntax.

- [Import hierarchy endpoint](#) on page 430

Use the `import hierarchy` endpoint to delete all existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.

- [Last class update endpoint](#) on page 431

Use the `last class update` endpoint to retrieve the time that classes were last updated from the primary server.

- [Update classes endpoint](#) on page 431

Use `update classes` endpoint to trigger the node classifier to update class and environment definitions from the primary server.

- [Validation endpoints](#) on page 432

Use validation endpoints to validate groups in the node classifier.

- [Node classifier errors](#) on page 435

Familiarize yourself with error responses to make working the node classifier service API easier.

Related information

[API index](#) on page 26

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

Forming node classifier requests

Requests to the node classifier API must be well-formed HTTP(S) requests.

By default, the node classifier service listens on port 4433 and all endpoints are relative to the `/classifier-api/` path. For example, the full URL for the `/v1/groups` endpoint on localhost would be `https://localhost:4433/classifier-api/v1/groups`.

If needed, you can change the port the classifier API listens on.

Related information

[Configuring the console](#) on page 171

After installing Puppet Enterprise, you can change product settings to customize the console's behavior. Many settings can be configured in the console itself.

Authenticating to the node classifier API

You need to authenticate requests to the node classifier API. You can do this using RBAC authentication tokens or with the list of allowed RBAC certificates.

Authentication token

You can make requests to the node classifier API using RBAC authentication tokens.

For detailed information about authentication tokens, see [token-based authentication](#).

In this example, we are using the `/groups` endpoint of the node classifier API to get a list of all groups that exist in the node classifier, along with their associated metadata. The example assumes that you have already generated a token with `puppet-access login`.

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"

curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The example above uses the X-Authentication header to supply the token information. In some cases, such as GitHub webhooks, you might need to supply the token in a token parameter. To supply the token in a token parameter, you would specify the request like this:

```
uri="https://$(puppet config print server):4433/classifier-api/v1/groups?
token=$(puppet-access show)"

curl --insecure "$uri"
```

Note: Supplying the token as a token parameter is not as secure as using the X-Authentication method.

Authenticating using an allowed certificate

You can also authenticate requests using a certificate listed in RBAC's certificate allowlist. The RBAC allowlist is located at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. If you edit this file, you must reload the `pe-console-services` service for your changes to take effect (`sudo service pe-console-services reload`). You can attach the certificate using the command line as demonstrated in the example cURL query below. You must have the allowed certificate name and the private key to run the script.

The following query returns a list of all groups that exist in the node classifier, along with their associated metadata. This query shows how to attach the allowlist certificate to authenticate the node classifier API.

In this query, the "allowlisted certname" needs to match a name in the file, `/etc/puppetlabs/console-services/rbac-certificate-allowlist`.

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
"$uri"
```

You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate specifically for use with the API.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Using pagination parameters

If you have a large number of groups, classes, nodes, node check-ins, or environments, then sending a GET request through the classifier API could return an excessively large number of items.

To limit the number of items returned, you can use the `limit` and `offset` parameters.

- `limit`: The value of the `limit` parameter limits how many items are returned in a response. The value must be a non-negative integer. If you specify a value other than a non-negative integer, you get a 400 Bad Request error.
- `offset`: The value of the `offset` parameter specifies the number of item that are skipped. For example, if you specify an offset of 20 with a limit of 10, as shown in the example below, the first 20 items are skipped and you get back item 21 through to item 30. The value must be a non-negative integer. If you specify a value other than a non-negative integer, you get a 400 Bad Request error.

The following example shows a request using the `limit` and `offset` parameters.

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups?
limit=10&offset=20"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key $key
"$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Groups endpoint

The `groups` endpoint is used to create, read, update, and delete groups.

A group belongs to an environment, applies classes (possibly with parameters), and matches nodes based on rules. Because groups are so central to the classification process, this endpoint is where most of the action is.

GET /v1/groups

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

Query parameters

The request accepts these parameters.

Parameter	Value
<code>inherited</code>	If set to any value besides 0 or <code>false</code> , the node group includes the classes, class parameters, configuration data, and variables that it inherits from its ancestors.

Response format

The response is a JSON array of node group objects, using these keys:

Key	Definition
<code>name</code>	The name of the node group (a string).
<code>id</code>	The node group's ID, which is a string containing a type-4 (random) UUID. The regular expression used to validate node group UUIDs is <code>[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}</code> .
<code>description</code>	An optional key containing an arbitrary string describing the node group.
<code>environment</code>	The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.
<code>environment_trumps</code>	This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.
<code>parent</code>	The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group, which is the root of the hierarchy. Note that the root group always has the lowest-possible random UUID, <code>00000000-0000-4000-8000-000000000000</code> .
<code>rule</code>	A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group.

Key	Definition
<code>classes</code>	An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the node group sets for that parameter (always a string).
<code>config_data</code>	An object similar to the <code>classes</code> object that specifies parameters that are applied to classes if the class is assigned in the classifier or in puppet code. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the group sets for that parameter (always a string). This feature is enabled/disabled via the <code>classifier::allow-config-data</code> setting. When set to false, this key is stripped from the payload.
<code>deleted</code>	An object similar to the <code>classes</code> object that shows which classes and class parameters set by the node group have since been deleted. If none of the node group's classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the <code>puppetlabs.classifier/deleted</code> key, whose value is a Boolean indicating whether the entire class has been deleted. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: <code>puppetlabs.classifier/deleted</code> , mapping to a boolean indicating whether the specific class parameter has been deleted; and <code>value</code> , mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the <code>classes</code> object).
<code>variables</code>	An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).
<code>last_edited</code>	The most recent time at which a user committed changes to a node group. This is a time stamp in ISO 8601 format, <code>YYYY-MM-DDTHH:MM:SSZ</code> .
<code>serial_number</code>	A number assigned to a node group. This number is incremented each time changes to a group are committed. <code>serial_number</code> is used to prevent conflicts when multiple users make changes to the same node group at the same time.

This example shows a node group object:

```
{
  "name": "Webservers",
  "id": "fc500c43-5065-469b-91fc-37ed0e500e81",
  "last_edited": "2018-02-20T02:36:17.776Z",
  "serial_number": 16,
  "environment": "production",
  "description": "This group captures configuration relevant to all web-
facing production webservers, regardless of location.",
  "parent": "00000000-0000-4000-8000-000000000000",
  "rule": ["and", ["~", ["trusted", "certname"], "www"],
    [">=", ["fact", "total_ram"], "512"]],
  "classes": {
    "apache": {
      "serveradmin": "bofh@travaglia.net",
      "keepalive_timeout": "5"
    }
  },
  "config_data": {
    "puppet_enterprise::profile::console": {"certname":
"console.example.com"},
    "puppet_enterprise::profile::puppetdb": {"listen_address": "0.0.0.0"}
  },
  "variables": {
    "ntp_servers": ["0.us.pool.ntp.org", "1.us.pool.ntp.org",
"2.us.pool.ntp.org"]
  }
}
```

This example shows a node group object that refers to some classes and parameters that have been deleted:

```
{
  "name": "Spaceship",
  "id": "fc500c43-5065-469b-91fc-37ed0e500e81",
  "last_edited": "2018-03-13T21:37:03.608Z",
  "serial_number": 42,
  "environment": "space",
  "parent": "00000000-0000-4000-8000-000000000000",
  "rule": ["=", ["fact", "is_spaceship"], "true"],
  "classes": {
    "payload": {
      "type": "cubesat",
      "count": "8",
      "mass": "10.64"
    },
    "rocket": {
      "stages": "3"
    }
  },
  "deleted": {
    "payload": {"puppetlabs.classifier/deleted": true},
    "rocket": {
      "puppetlabs.classifier/deleted": false,
      "stages": {
        "puppetlabs.classifier/deleted": true,
        "value": "3"
      }
    }
  },
  "variables": {}
}
```


The entire payload class has been deleted, because its deleted parameters object's `puppetlabs.classifier/deleted` key maps to `true`, in contrast to the `rocket` class, which has had only its `stages` parameter deleted.

Rule condition grammar

Nodes can be classified into groups using rules. This example shows how rule conditions must be structured:

```

condition  : [ {bool} {condition}+ ] | [ "not" {condition} ] |
{operation}
    bool    : "and" | "or"
    operation : [ {operator} {fact-path} {value} ]
    operator : "=" | "~" | ">" | ">=" | "<" | "<="
    fact-path : {field-name} | [ {path-type} {field-name} {path-
component}+ ]
    path-type : "trusted" | "fact"
    path-component : field-name | number
    field-name : string

```

For the regex operator `"~"`, the value is interpreted as a Java regular expression. Literal backslashes must be used to escape regex characters in order to match those characters in the fact value.

For the numeric comparison operators (`">"`, `">="`, `"<"`, and `"<="`), the fact value (which is always a string) is coerced to a number (either integral or floating-point). If the value can't be coerced to a number, the numeric operation evaluates to false.

For the fact path, the rule can be either a string representing a top level field (the only current meaningful value here would be `"name"` representing the node name) or a list of strings and indices that represent looking up a field in a nested data structure. When passing a list of strings or indices, the first and second entries in the list must be strings. Subsequent entries can be indices.

Regular facts start with `"fact"` (for example, `["fact", "architecture"]`) and trusted facts start with `"trusted"` (for example, `["trusted", "certname"]`).

Error responses

`serial_number`

If you commit a node group with a `serial_number` that an API call has previously assigned, the service returns a 409 Conflict response.

POST /v1/groups

Use the `/v1/groups` endpoint to create a new node group without specifying its ID. When you use this endpoint, the node classifier service randomly generates an ID.

Request format

The request body must be a JSON object describing the node group to be created. The request uses these keys:

Key	Definition
<code>name</code>	The name of the node group (a string).
<code>environment</code>	The name of the node group's environment. This key is optional; if it's not provided, the default environment (<code>production</code>) is used.
<code>environment_trumps</code>	Whether this node group's environment should override those of other node groups at classification-time. This key is optional; if it's not provided, the default value of <code>false</code> is used.

Key	Definition
<code>description</code>	A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.
<code>parent</code>	The ID of the node group's parent (required).
<code>rule</code>	The condition that must be satisfied for a node to be classified into this node group
<code>variables</code>	An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The <code>variables</code> key is optional, and if a node group does not define any top-level variables then it may be omitted.
<code>classes</code>	An object that defines the classes to be used by nodes in the node group. The <code>classes</code> key is required, and at minimum should be an empty object (<code>{}</code>). The <code>classes</code> key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The <code>classes</code> key is <i>not</i> optional; if it is missing, the service returns a 400Bad Request response.
<code>config_data</code>	<p>An optional object that defines the class parameters to be used by nodes in the group. Its structure is the same as the <code>classes</code> object. If you use a <code>config_data</code> key but provide only the class, like <code>"config_data": { "qux": {} }</code>, no configuration data is stored.</p> <p>Note: This feature is enabled with the <code>classifier::allow-config-data</code> setting. When set to false, the presence of this key in the payload results in a 400 response.</p>

Response format

If the node group was successfully created, the service returns a 303 See Other response, with the path to retrieve the created node group in the "location" header of the response.

Error responses

Responses and keys returned for create group requests depend on the type of error.

`schema-violation`

If any of the required keys are missing or the values of any of the defined keys do not match the required type, the service returns a `400 Bad Request` response using these keys:

Key	Definition
kind	"schema-violation"
details	<p>An object that contains three keys:</p> <ul style="list-style-type: none"> • <code>submitted</code> — Describes the submitted object. • <code>schema</code> — Describes the schema that object is expected to conform to. • <code>error</code> — Describes how the submitted object failed to conform to the schema.

malformed-request

If the request's body could not be parsed as JSON, the service returns a `400 Bad Request` response using these keys:

Key	Definition
kind	"malformed-request"
details	<p>An object that contains two keys:</p> <ul style="list-style-type: none"> • <code>body</code> — Holds the request body that was received. • <code>error</code> — Describes how the submitted object failed to conform to the schema.

uniqueness-violation

If your attempt to create the node group violates uniqueness constraints (such as the constraint that each node group name must be unique within its environment), the service returns a `422 Unprocessable Entity` response using these keys:

Key	Definition
kind	"uniqueness-violation"
msg	Describes which fields of the node group caused the constraint to be violated, along with their values.
details	<p>An object that contains two keys:</p> <ul style="list-style-type: none"> • <code>conflict</code> — An object whose keys are the fields of the node group that violated the constraint and whose values are the corresponding field values. • <code>constraintName</code> — The name of the database constraint that was violated.

missing-referents

If classes or class parameters defined by the node group, or inherited by the node group from its parent, do not exist in the submitted node group's environment, the service returns a 422 `Unprocessable Entity` response. In both cases the response object uses these keys:

Key	Definition
kind	"missing-referents"
msg	Describes the error and lists the missing classes or parameters.
details	<p>An array of objects, where each object describes a single missing referent, and has the following keys:</p> <ul style="list-style-type: none"> • <code>kind</code> — "missing-class" or "missing-parameter", depending on whether the entire class doesn't exist, or the class just doesn't have the parameter. • <code>missing</code> — The name of the missing class or class parameter. • <code>environment</code> — The environment that the class or parameter is missing from; that is, the environment of the node group where the error was encountered. • <code>group</code> — The name of the node group where the error was encountered. Due to inheritance, this might not be the group where the parameter was defined. • <code>defined_by</code> — The name of the node group that defines the class or parameter.

missing-parent

If the parent of the node group does not exist, the service returns a 422 `Unprocessable Entity` response. The response object uses these keys:

Key	Definition
kind	"missing-parent"
msg	Shows the parent UUID that did not exist.
details	The full submitted node group.

inheritance-cycle

If the request causes an inheritance cycle, the service returns a 422 `Unprocessable Entity` response. The response object uses these keys:

Key	Definition
kind	"inheritance-cycle"
details	An array of node group objects that includes each node group involved in the cycle

Key	Definition
msg	A shortened description of the cycle, including a list of the node group names with each followed by its parent until the first node group is repeated.

GET /v1/groups/<id>

Use the /v1/groups/<id> endpoint to retrieve a node group with the given ID.

Response format

The response is a JSON array of node group objects, using these keys:

Key	Definition
name	The name of the node group (a string).
id	The node group's ID, which is a string containing a type-4 (random) UUID. The regular expression used to validate node group UUIDs is <code>[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}</code> .
description	An optional key containing an arbitrary string describing the node group.
environment	The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.
environment_trumps	This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.
parent	The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group, which is the root of the hierarchy. Note that the root group always has the lowest-possible random UUID, 00000000-0000-4000-8000-000000000000.
rule	A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group.

Key	Definition
<code>classes</code>	An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the node group sets for that parameter (always a string).
<code>deleted</code>	An object similar to the <code>classes</code> object that shows which classes and class parameters set by the node group have since been deleted. If none of the node group's classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the <code>puppetlabs.classifier/deleted</code> key, whose value is a Boolean indicating whether the entire class has been deleted. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: <code>puppetlabs.classifier/deleted</code> , mapping to a boolean indicating whether the specific class parameter has been deleted; and <code>value</code> , mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the <code>classes</code> object).
<code>variables</code>	An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).
<code>last_edited</code>	The most recent time at which a user committed changes to a node group. This is a time stamp in ISO 8601 format, <code>YYYY-MM-DDTHH:MM:SSZ</code> .
<code>serial_number</code>	A number assigned to a node group. This number is incremented each time changes to a group are committed. <code>serial_number</code> is used to prevent conflicts when multiple users make changes to the same node group at the same time.

Error responses

If the node group with the given ID cannot be found, the service returns 404 `Not Found` and `malformed-uuid` responses. The body includes a generic 404 error response as described in the errors documentation.

`serial_number`

If you commit a node group with a `serial_number` that an API call has previously assigned, the service returns a 409 `Conflict` response.

Related information

[Node classifier errors](#) on page 435

Familiarize yourself with error responses to make working the node classifier service API easier.

PUT /v1/groups/<id>

Use the `/v1/groups/<id>` to create a node group with the given ID.



CAUTION: Any existing node group with the given ID is overwritten.

It is possible to overwrite an existing node group with a new node group definition that contains deleted classes or parameters.

Important: You must supply a valid type-4 (random) UUID for the ID.

Request format

The request body must be a JSON object describing the node group to be created. The request uses these keys:

Key	Definition
<code>name</code>	The name of the node group (a string).
<code>environment</code>	The name of the node group's environment. This key is optional; if it's not provided, the default environment (<code>production</code>) is used.
<code>environment_trumps</code>	Whether this node group's environment should override those of other node groups at classification-time. This key is optional; if it's not provided, the default value of <code>false</code> is used.
<code>description</code>	A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.
<code>parent</code>	The ID of the node group's parent (required).
<code>rule</code>	The condition that must be satisfied for a node to be classified into this node group
<code>variables</code>	An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The <code>variables</code> key is optional, and if a node group does not define any top-level variables then it may be omitted.

Key	Definition
<code>classes</code>	An object that defines the classes to be used by nodes in the node group. The <code>classes</code> key is required, and at minimum should be an empty object (<code>{}</code>). The <code>classes</code> key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The <code>classes</code> key is <i>not</i> optional; if it is missing, the service returns a 400Bad Request response.

Response format

If the node group is successfully created, the service returns a 201 Created response, with the node group object (in JSON) as the body. If the node group already exists and is identical to the submitted node group, then the service takes no action and returns a 200 OK response, again with the node group object as the body.

Error responses

If the requested node group object contains the `id` key, and its value differs from the UUID specified in the request's path, the service returns a 400 Bad Request response.

The response object uses these keys:

Key	Definition
<code>kind</code>	"conflicting-ids"
<code>details</code>	An object that contains two keys: <ul style="list-style-type: none"> <code>submitted</code> — Contains the ID submitted in the request body. <code>fromUrl</code> — Contains the ID taken from the request URL.

In addition, this operation can produce the general `malformed-error` response and any response that could also be generated by the POST group creation endpoint.

POST /v1/groups/<id>

Use the `/v1/groups/<id>` endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Request format

The request body must be JSON object describing the delta to be applied to the node group.

The `classes`, `config_data`, `variables`, and `rule` keys of the delta are merged with the node group, and then any keys of the resulting object that have a null value are deleted. This allows you to remove classes, class parameters, configuration data, variables, or the rule from the node group by setting them to null in the delta.

If the delta has a rule key that's set to a new value or nil, it's updated wholesale or removed from the group accordingly.

The name, environment, description, and parent keys, if present in the delta, replace the old values wholesale with their values.

The `serial_number` key is optional. If you update a node group and provide the `serial_number` in the payload, and the `serial_number` is not the current one for that group, the service returns a 409 Conflict response. To bypass this check, omit the `serial_number`.

Note that the root group's rule cannot be edited; any attempts to do so raise a 422 Unprocessable Entity response.

In the following examples, a delta is merged with a node group to update the group.

Node group:

```
{
  "name": "Webservers",
  "id": "58463036-0efa-4365-b367-b5401c0711d3",
  "environment": "staging",
  "parent": "00000000-0000-4000-8000-000000000000",
  "rule": [ "~", [ "trusted", "certname", "www" ],
  "classes": {
    "apache": {
      "serveradmin": "bofh@travaglia.net",
      "keepalive_timeout": 5
    },
    "ssl": {
      "keystore": "/etc/ssl/keystore"
    }
  },
  "variables": {
    "ntp_servers": [ "0.us.pool.ntp.org", "1.us.pool.ntp.org",
    "2.us.pool.ntp.org" ]
  }
}
```

Delta:

```
{
  "name": "Production Webservers",
  "id": "58463036-0efa-4365-b367-b5401c0711d3",
  "environment": "production",
  "parent": "01522c99-627c-4a07-b28e-a25dd563d756",
  "classes": {
    "apache": {
      "serveradmin": "roy@reynholm.co.uk",
      "keepalive_timeout": null
    },
    "ssl": null
  },
  "variables": {
    "dns_servers": [ "dns.reynholm.co.uk" ]
  }
}
```

Updated group:

```
{
  "name": "Production Webservers",
  "id": "58463036-0efa-4365-b367-b5401c0711d3",
  "environment": "production",
```

```

"parent": "01522c99-627c-4a07-b28e-a25dd563d756",
"rule": [ "~", [ "trusted", "certname", "www" ],
"classes": {
  "apache": {
    "serveradmin": "roy@reynholm.co.uk"
  }
},
"variables": {
  "ntp_servers": [ "0.us.pool.ntp.org", "1.us.pool.ntp.org",
"2.us.pool.ntp.org" ],
  "dns_servers": [ "dns.reynholm.co.uk" ]
}
}

```

Note that the `ssl` class was deleted because its entire object was mapped to null, whereas for the `apache` class only the `keepalive_timeout` parameter was deleted.

Deleted classes and class parameters

If the node group definition contains classes and parameters that have been deleted it is still possible to update the node group with those parameters and classes. Updates that don't increase the number of errors associated with a node group are allowed.

Error responses

The response object uses these keys:

Key	Definition
kind	"conflicting-ids"
details	<p>An object that contains two keys:</p> <ul style="list-style-type: none"> <code>submitted</code> — Contains the ID submitted in the request body. <code>fromUrl</code> — Contains the ID taken from the request URL.

If the requested node group object contains the `id` key, and its value differs from the UUID specified in the request's path, the service returns a 400 `Bad Request` response.

In addition, this operation can produce the general `malformed-error` response and any response that could also be generated by the POST group creation endpoint.

422 responses to POST requests can include errors that were caused by the node group's children, but a node group being created with a PUT request cannot have any children.

DELETE /v1/groups/<id>

Use the `/v1/groups/<id>` endpoint to delete the node group with the given ID.

Response format

If the delete operation is successful, the sever returns a 204 `No Content` response.

Error responses

In addition to the general `malformed-uuid` response, if the node group with the given ID does not exist, the service returns a 404 `Not Found` response, as described in the errors documentation.

children-present

The service returns a 422 `Unprocessable Entity` and rejects the delete request if the node group that is being deleted has children.

The response object uses these keys:

Key	Definition
kind	"children-present"
msg	Explains why the delete was rejected and names the children.
details	Contains the node group in question along with all of its children.

Related information

[Node classifier errors](#) on page 435

Familiarize yourself with error responses to make working the node classifier service API easier.

POST /v1/groups/<id>/pin

Use the `/v1/groups/<id>/pin` endpoint to pin nodes to the group with the given ID.

Request format

You can provide the names of the nodes to pin in two ways.

- As the value of the `nodes` query parameter. For multiple nodes, use a comma-separated list format.

For example:

```
POST /v1/groups/58463036-0efa-4365-b367-b5401c0711d3/pin?nodes=foo%2Cbar%2Cbaz
```

This request pins the nodes `foo`, `bar`, and `baz` to the group.

- In the body of a request. In the `nodes` field of a JSON object, specify the node name as the value. For multiple nodes, use a JSON array.

For example:

```
{ "nodes": [ "foo", "bar", "baz" ] }
```

This request pins the nodes `foo`, `bar`, and `baz` to the group.

It's easier to use the query parameter method. However, if you want to affect a large number of nodes at once, the query string might get truncated. Strings are truncated if they exceed 8,000 characters. In such cases, use the second method. The request body in the second method is allowed to be many megabytes in size.

Response format

If the pin is successful, the service returns a 204 `No Content` response with an empty body.

Error responses

This endpoint uses the following error responses.

If you don't supply the `nodes` query parameter or a request body, the service returns a 400 `Malformed Request` response, using these keys:

Key	Definition
kind	"missing-parameters"
msg	Explains the missing <code>nodes</code> query parameter.

If you supply a request body that is not valid JSON, the service returns a 400 `Malformed Request` response. The response object uses these keys:

Key	Definition
<code>kind</code>	"malformed-request"
<code>details</code>	An object with a <code>body</code> key containing the body as received by the service, and an <code>error</code> field containing a string describing the error encountered when trying to parse the request's body.

If the request's body is valid JSON, but the payload is not an object with just the `nodes` field and no other fields, the service returns a 400 `Malformed Request` response. The response object uses these keys:

Key	Definition
<code>kind</code>	"schema-violation"
<code>msg</code>	Describes the difference between what was submitted and the required format.

POST /v1/groups/<id>/unpin

Use the `/v1/groups/<id>/unpin` endpoint to unpin nodes from the group with the given ID.

Note: Nodes not actually pinned to the group can be specified without resulting in an error.

Request format

You can provide the names of the nodes to pin in two ways.

- As the value of the `nodes` query parameter. For multiple nodes, use a comma-separated list format.

For example:

```
POST /v1/groups/58463036-0efa-4365-b367-b5401c0711d3/unpin?nodes=foo%2Cbar%2Cbaz
```

This request unpins the nodes `foo`, `bar`, and `baz` from the group. If any of the specified nodes were not pinned to the group, they are ignored.

- In the body of a request. In the `nodes` field of a JSON object, specify the node name as the value. For multiple nodes, use a JSON array.

For example:

```
{"nodes": ["foo", "bar", "baz"]}
```

This request unpins the nodes `foo`, `bar`, and `baz` from the group. If any of the specified nodes were not pinned to the group, they are ignored.

It's easier to use the query parameter method. However, if you want to affect a large number of nodes at once, the query string might get truncated. Strings are truncated if they exceed 8,000 characters. In such cases, use the second method. The request body in the second method is allowed to be many megabytes in size.

Response format

If the unpin is successful, the service returns a 204 `No Content` response with an empty body.

Error responses

If you don't supply the `nodes` query parameter or a request body, the service returns a 400 `Malformed Request` response. The response object uses these keys:

Key	Definition
kind	"missing-parameters"
msg	Explains the missing nodes query parameter.

If a request body is supplied but it is not valid JSON, the service returns a 400 `Malformed Request` response. The response object uses these keys:

Key	Definition
kind	"malformed-request"
details	An object with a <code>body</code> key containing the body as received by the service, and an <code>error</code> field containing a string describing the error encountered when trying to parse the request's body.

If the request's body is valid JSON, but the payload is not an object with just the `nodes` field and no other fields, the service returns a 400 `Malformed Request` response. The response object uses these keys:

Key	Definition
kind	"schema-violation"
msg	Describes the difference between what was submitted and the required format.

GET /v1/groups/:id/rules

Use `/v1/groups/:id/rules` to resolve the rules for the requested group and then translate those rules to work with the nodes and inventory endpoints in PuppetDB.

Response format

A successful response includes these keys:

rule

The rules for the group in classifier format.

rule_with_inherited

The inherited rules (including the rules for this group) in classifier format

translated

An object containing two children, each of the inherited rules translated into a different format.

nodes_query_format

The optimized translated inherited group in the format that works with the nodes endpoint in PuppetDB.

inventory_query_format

The optimized translated inherited group in the format that works with the inventory endpoint in PuppetDB.

```
{
  "rule": [
    "=",
    [
      "fact",
      "is_spaceship"
    ],
    "true"
  ],
  "rule_with_inherited": [
    "and",
```

```

[
  "=",
  [
    "fact",
    "is_spaceship"
  ],
  "true"
],
[
  "~",
  "name",
  ".*"
]
],
"translated": {
  "nodes_query_format": [
    "or",
    [
      "=",
      [
        "fact",
        "is_spaceship"
      ],
      "true"
    ],
    [
      "=",
      [
        "fact",
        "is_spaceship",
        true
      ]
    ]
  ],
  "inventory_query_format": [
    "or",
    [
      "=",
      "facts.is_spaceship",
      "true"
    ],
    [
      "=",
      "facts.is_spaceship",
      true
    ]
  ]
}
}

```

Error responses

In addition to the general malformed-uuid error response, if the group with the given ID cannot be found, a 404 Not Found response is returned.

Related information

[Error response description](#) on page 436

Errors from the node classifier service are JSON responses.

Groups endpoint examples

Use example requests to better understand how to work with groups in the node classifier API.

These requests assume the following configuration:

- The Puppet primary server is running on `puppetlabs-nc.example.vm` and can allow certificates to enable URL requests.
- Port 4433 is open.

Create a group called My Nodes

This request uses `POST /v1/groups` to create a group called *My Nodes*.

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
data='{
  "name": "My Nodes",
  "parent": "00000000-0000-4000-8000-000000000000",
  "environment": "production",
  "classes": {}
}'

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
--request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Related information

[POST /v1/groups](#) on page 393

Use the `/v1/groups` endpoint to create a new node group without specifying its ID. When you use this endpoint, the node classifier service randomly generates an ID.

Get the group ID of My Nodes

This request uses the groups endpoint to get details about groups.

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
"$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The response is a JSON file containing details about all groups, including the *My Nodes* group ID.

```
{
  "environment_trumps": false,
  "parent": "00000000-0000-4000-8000-000000000000",
  "name": "My Nodes",
  "variables": {},
  "id": "085e2797-32f3-4920-9412-8e9decf4ef65",
  "environment": "production",
  "classes": {}
}
```

Related information

[Groups endpoint](#) on page 389

The groups endpoint is used to create, read, update, and delete groups.

Pin a node to the My Nodes group

This request uses POST `/v1/groups/<id>` to pin the node *example-to-pin.example.vm* to *My Groups* using the group ID retrieved in the previous step.

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin"
data='{ "nodes": [ "example-to-pin.example.vm" ] }'

curl --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Related information

[POST /v1/groups/<id>](#) on page 400

Use the `/v1/groups/<id>` endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Pin a second node to the My Nodes group

This request uses POST `/v1/groups/<id>` to pin a second node *example-to-pin-2.example.vm* to *My Groups*.

Note: You must supply all the nodes to pin to the group. For example, this request includes both *example-to-pin.example.vm* and *example-to-pin-2.example.vm*.

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin"
data='{ "nodes": [ "example-to-pin.example.vm", "example-to-pin-2.example.vm" ] }'

curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Related information

[POST /v1/groups/<id>](#) on page 400

Use the `/v1/groups/<id>` endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Unpin a node from the My Nodes group

This request uses POST `/v1/groups/<id>` to unpin the node *example-to-unpin.example.vm* from *My Groups*.

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups/a02ee4a7-8ecl-44ec-99a6-ef362596fd6e/unpin"
data='{ "nodes": [ "example-to-unpin" ] }'

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key" --request POST "$uri" --data "$data"
```


See [Usage notes for curl examples](#) for information about forming curl commands.

Add a class and parameters to the My Nodes group

This request uses POST `/v1/groups/<id>` to specify the *apache* class and parameters for *My Groups*.

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65"
data='{ "classes":
      { "apache": {
          "serveradmin": "bob@example.com",
          "keepalive_timeout": null}
      }
    }'
```

```
curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Related information

[POST /v1/groups/<id>](#) on page 400

Use the `/v1/groups/<id>` endpoint to update the name, environment, parent, rule, classes, class parameters, configuration data, and variables of the node group with the given ID by submitting a node group delta.

Classes endpoint

Use the `classes` endpoints to retrieve lists of classes, including classes with specific environments. The output from this endpoint is especially useful for creating new node groups, which usually contain a reference to one or more classes.

The node classifier gets its information about classes from Puppet, so don't use this endpoint to create, update, or delete classes.

GET /v1/classes

Use the `/v1/classes` endpoint to retrieve a list of all classes known to the node classifier.

Note: All other operations on classes require using the environment-specific endpoints.

All `/classes` endpoints return the classes *currently known* to the node classifier, which retrieves them periodically from the primary server. To force an update, use the `update_classes` endpoint. To determine when classes were last retrieved from the primary server, use the `last_class_update` endpoint.

Response format

The response is a JSON array of class objects. Each class object contains these keys:

Key	Definition
name	The name of the class (a string).
environment	The name of the environment that this class exists in. Note that the same class can exist in different environments and can have different parameters in each environment.

Key	Definition
parameters	An object describing the parameters and default parameter values for the class. The keys of this object are the parameter names (strings). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or null. If the value is null, the parameter is required.

This is an example of a class object:

```
{
  "name": "apache",
  "environment": "production",
  "parameters": {
    "default_mods": true,
    "default_vhost": true,
    ...
  }
}
```

GET /v1/environments/<environment>/classes

Use the `/v1/environments/\<environment\>/classes` to retrieve a list of all classes known to the node classifier within the given environment.

Response format

The response is a JSON array of class objects. Each class object contains these keys:

Key	Definition
name	The name of the class (a string).
environment	The name of the environment that this class exists in. Note that the same class can exist in different environments and can have different parameters in each environment.
parameters	An object describing the parameters and default parameter values for the class. The keys of this object are the parameter names (strings). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or null. If the value is null, the parameter is required.

Error responses

This request does not produce error responses.

GET /v1/environments/<environment>/classes/<name>

Use the `/v1/environments/\<environment\>/classes/\<name\>` endpoint to retrieve the class with the given name in the given environment.

Response format

If the class exists, the response is a JSON array of class objects. Each class object contains these keys:

Key	Definition
<code>name</code>	The name of the class (a string).
<code>environment</code>	The name of the environment that this class exists in. Note that the same class can exist in different environments and can have different parameters in each environment.
<code>parameters</code>	An object describing the parameters and default parameter values for the class. The keys of this object are the parameter names (strings). Each value is the default value for the associated parameter as a string, boolean, number, structured value, or <code>null</code> . If the value is <code>null</code> , the parameter is required.

Error responses

If the class with the given name cannot be found, the service returns a 404 `Not Found` response with an empty body.

Classification endpoint

The `classification` endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

POST /v1/classified/nodes/<name>

Use the `/v1/classified/nodes/\<name\>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

Request format

The request body can contain a JSON object describing the facts and trusted facts of the node to be classified. The object can have these keys:

Key	Definition
<code>fact</code>	The regular, non-trusted facts of the node. The value of this key is a further object, whose keys are fact names, and whose values are the fact values. Fact values can be a string, number, boolean, array, or object.
<code>trusted</code>	The trusted facts of the node. The values of this key are subject to the same restrictions as those on the value of the <code>fact</code> key.

Response format

The response is a JSON object describing the node post-classification, using these keys:

Key	Definition
<code>name</code>	The name of the node (a string).
<code>groups</code>	An array of the group-ids (strings) that this node was classified into.

Key	Definition
environment	The name of the environment that this node uses, which is taken from the node groups the node was classified into.
classes	An object where the keys are class names and the values are objects that map parameter names to values.
parameters	An object where the keys are top-level variable names and the values are the values assigned to those variables.

This is an example of a response from this endpoint:

```
{
  "name": "foo.example.com",
  "groups": [ "9c0c7d07-a199-48b7-9999-3cdf7654e0bf", "96d1a058-225d-48e2-
    ala8-80819d31751d" ],
  "environment": "staging",
  "parameters": {},
  "classes": {
    "apache": {
      "keepalive_timeout": 30,
      "log_level": "notice"
    }
  }
}
```

Error responses

If the node is classified into multiple node groups that define conflicting classifications for the node, the service returns a 500 `Server Error` response.

The body of this response contains the usual JSON error object described in the errors documentation.

The `kind` key of the error is "classification-conflict", the `msg` describes generally why this happens, and the `details` key contains an object that describes the specific conflicts encountered.

The details object can have these keys:

Key	Definition
environment	Maps directly to an array of value detail objects (described below).
variables	Contains an object with a key for each conflicting variable, whose values are an array of value detail objects.
classes	Contains an object with a key for each class that had conflicting parameter definitions, whose values are further objects that describe the conflicts for that class's parameters.

A value details object describes one of the conflicting values defined for the environment, a variable, or a class parameter. Each object contains these keys:

Key	Definition
value	The defined value, which is a string for environment and class parameters, but for a variable can be any JSON value.

Key	Definition
from	The node group that the node was classified into that caused this value to be added to the node's classification. This group cannot define the value, because it can be inherited from an ancestor of this group.
defined_by	The node group that actually defined this value. This is often the from group, but could instead be an ancestor of that group.

This example shows a classification conflict error object with node groups truncated for clarity:

```
{
  "kind": "classification-conflict",
  "msg": "The node was classified into multiple unrelated node groups that
defined conflicting class parameters or top-level variables. See `details`
for a list of the specific conflicts.",
  "details": {
    "classes": {
      "songColors": {
        "blue": [
          {
            "value": "Blue Suede Shoes",
            "from": {
              "name": "Elvis Presley",
              "classes": {},
              "rule": ["=", "nodename", "the-node"],
              ...
            },
            "defined_by": {
              "name": "Carl Perkins",
              "classes": {"songColors": {"blue": "Blue Suede Shoes"}},
              "rule": ["not", ["=", "nodename", "the-node"]],
              ...
            }
          },
          {
            "value": "Since You've Been Gone",
            "from": {
              "name": "Aretha Franklin",
              "classes": {"songColors": {"blue": "Since You've Been Gone"}},
              ...
            },
            "defined_by": {
              "name": "Aretha Franklin",
              "classes": {"songColors": {"blue": "Since You've Been Gone"}},
              ...
            }
          }
        ]
      }
    }
  }
}
```

In this example, the conflicting "Blue Suede Shoes" value was included in the classification because the node matched the "Elvis Presley" node group (because that is the value of the "from" key), but that node group doesn't define the "Blue Suede Shoes" value. That value is defined by the "Carl Perkins" node group, which is an ancestor of the "Elvis Presley" node group, causing the latter to inherit the value from the former. The other conflicting value, "Since You've Been Gone", is defined by the same node group that the node matched.

Related information

[Node classifier errors](#) on page 435

Familiarize yourself with error responses to make working the node classifier service API easier.

POST /v1/classified/nodes/<name>/explanation

Use the `/v1/classified/nodes/<name>/explanation` endpoint to retrieve an explanation of how a node is classified by submitting its facts.

Request format

The body of the request must be a JSON object describing the node's facts. This object uses these keys:

Key	Definition
fact	Describes the regular (non-trusted) facts of the node. Its value must be a further object whose keys are fact names, and whose values are the corresponding fact values. Structured facts can be included here; structured fact values are further objects or arrays.
trusted	Optional key that describes the trusted facts of the node. Its value has exactly the same format as the <code>fact</code> key's value.
name	The node's name from the request's URL is merged into this object under this key.

This is an example of a valid request body:

```
{
  "fact": {
    "ear-tips": "pointed",
    "eyebrow pitch": "40",
    "hair": "dark",
    "resting bpm": "120",
    "blood oxygen transporter": "hemocyanin",
    "anterior tricuspid": "2",
    "appendices": "1",
    "spunk": "10"
  }
}
```

Response format

The response is a JSON object that describes how the node would be classified.

- If the node would be successfully classified, this object contains the final classification.
- If the classification would fail due to conflicts, this object contains a description of the conflicts.

This response is intended to provide insight into the entire classification process, so that if a node isn't classified as expected, you can trace the deviation.

Classification proceeds in this order:

1. All node group rules are tested on the node's facts and name, and groups that don't match the node are culled, leaving the matching groups.
2. Inheritance relations are used to further cull the matching groups, by removing any matching node group that has a descendant that is also a matching node group. Those node groups that are left over are *leaf groups*.
3. Each leaf group is transformed into its inherited classification by adding all the inherited values from its ancestors.

4. All of the inherited classifications and individual node classifications are inspected for conflicts. A conflict occurs whenever two inherited classifications define different values for the environment, a class parameter, or a top-level variable.
5. Any individual node classification, including classes, class parameters, configuration data, and variables, is added.
6. Individual node classification is applied to the group classification, forming the final classification, which is then returned to the client.

The JSON object returned by this endpoint uses these keys:

Key	Definition
<code>match_explanations</code>	Corresponds to step 1 of classification, finding the matching node groups. This key's value is an explanation object just like those found in node check-ins, which maps between a matching group's ID and an explained condition object that demonstrates why the node matched that group's rule.
<code>leaf_groups</code>	Corresponds to step 2 of classification, finding the leaves. This key's value is an array of the leaf groups (that is, those groups that are not related to any of the other matching groups).
<code>inherited_classifications</code>	Corresponds to step 3 of classification, adding inherited values. This key's value is an object mapping from a leaf group's ID to the classification values provided by that group (after inheritance).
<code>conflicts</code>	Corresponds to step 4 of classification. This key is present only if there are conflicts between the inherited classifications. Its value is similar to a classification object, but wherever there was a conflict there's an array of conflict details instead of a single classification value. Each of these details is an object with three keys: <code>value</code> , <code>from</code> , and <code>defined_by</code> . The <code>value</code> key is a conflicting value, the <code>from</code> key is the group whose inherited classification provided this value, and the <code>defined_by</code> key is the group that actually defined the value (which can be an ancestor of the <code>from</code> group).
<code>individual_classification</code>	Corresponds to step 5 of classification. Its value includes classes, class parameters, configuration data, and variables applied directly during this stage.
<code>final_classification</code>	Corresponds to step 6 of classification, present only if there are no conflicts between the inherited classifications. Its value is the result of merging all individual node classification and group classification.
<code>node_as_received</code>	The submitted node object as received by the service, after adding the name and, if not supplied by the client, an empty <code>trusted</code> object. This key does not correspond to any of the classification steps.
<code>classification_source</code>	An annotated version of the classification that has the environment and every class parameter and variable replaced with an "annotated value" object. This key does not correspond to any of the classification steps.

This example shows a response the endpoint could return in the case of a successful classification:

```
{
  "node_as_received": {
    "name": "Tuvok",
    "trusted": {},
    "fact": {
      "ear-tips": "pointed",
      "eyebrow pitch": "30",
      "blood oxygen transporter": "hemocyanin",
      "anterior tricuspid": "2",
      "hair": "dark",
      "resting bpm": "200",
      "appendices": "0",
      "spunk": "0"
    }
  },
  "match_explanations": {
    "00000000-0000-4000-8000-000000000000": {
      "value": true,
      "form": [{"~", {"path": "name", "value": "Tuvok"}, ".*"}]
    },
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "value": true,
      "form": ["and",
        {
          "value": true,
          "form": [">=", {"path": ["fact", "eyebrow pitch"], "value":
"30"}, "25"]
        },
        {
          "value": true,
          "form": ["=", {"path": ["fact", "ear-tips"], "value":
"pointed"}, "pointed"]
        },
        {
          "value": true,
          "form": ["=", {"path": ["fact", "hair"], "value": "dark"},
"dark"]
        },
        {
          "value": true,
          "form": [">=", {"path": ["fact", "resting bpm"], "value":
"200"}, "100"]
        },
        {
          "value": true,
          "form": ["=",
            {
              "path": ["fact", "blood oxygen transporter"],
              "value": "hemocyanin"
            },
            "hemocyanin"
          ]
        }
      ]
    }
  },
  "leaf_groups": {
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "name": "Vulcans",
      "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
      "parent": "00000000-0000-4000-8000-000000000000",

```



```

      "rule": ["and", [ ">=", ["fact", "eyebrow pitch"], "25"],
                     [ "=", ["fact", "ear-tips"], "pointed"],
                     [ "=", ["fact", "hair"], "dark"],
                     [ ">=", ["fact", "resting bpm"], "100"],
                     [ "=", ["fact", "blood oxygen transporter"],
"hemocyanin"]
    ],
    "environment": "alpha-quadrant",
    "variables": {},
    "classes": {
      "emotion": {"importance": "ignored"},
      "logic": {"importance": "primary"}
    },
    "config_data": {
      "USS::Voyager": {"designation": "subsequent"}
    }
  },
  "inherited_classifications": {
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "environment": "alpha-quadrant",
      "variables": {},
      "classes": {
        "logic": {"importance": "primary"},
        "emotion": {"importance": "ignored"}
      },
      "config_data": {
        "USS::Enterprise": {"designation": "original"},
        "USS::Voyager": {"designation": "subsequent"}
      }
    }
  },
  "individual_classification": {
    "classes": {
      "emotion": {
        "importance": "secondary"
      }
    },
    "variables": {
      "full_name": "S'chn T'gai Spock"
    }
  },
  "final_classification": {
    "environment": "alpha-quadrant",
    "variables": {
      "full_name": "S'chn T'gai Spock"
    },
    "classes": {
      "logic": {"importance": "primary"},
      "emotion": {"importance": "secondary"}
    },
    "config_data": {
      "USS::Enterprise": {"designation": "original"},
      "USS::Voyager": {"designation": "subsequent"}
    }
  },
  "classification_sources": {
    "environment": {
      "value": "alpha-quadrant",
      "sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
    },
    "variables": {},
    "classes": {
      "emotion": {

```

```

      "puppetlabs.classifier/sources":
      ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"],
      "importance": {
        "value": "secondary",
        "sources": ["node"]
      }
    },
    "logic": {
      "puppetlabs.classifier/sources":
      ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"],
      "importance": {
        "value": "primary",
        "sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
      }
    },
    "config_data": {
      "USS::Enterprise": {
        "designation": {
          "value": "original",
          "sources": ["00000000-0000-4000-8000-000000000000"]
        }
      },
      "USS::Voyager": {
        "designation": {
          "value": "subsequent",
          "sources": ["8aeeb640-8dca-4b99-9c40-3b75de6579c2"]
        }
      }
    }
  }
}

```

This example shows a response that resulted from conflicts:

```

{
  "node_as_received": {
    "name": "Spock",
    "trusted": {},
    "fact": {
      "ear-tips": "pointed",
      "eyebrow pitch": "40",
      "blood oxygen transporter": "hemocyanin",
      "anterior tricuspid": "2",
      "hair": "dark",
      "resting bpm": "120",
      "appendices": "1",
      "spunk": "10"
    }
  },
  "match_explanations": {
    "00000000-0000-4000-8000-000000000000": {
      "value": true,
      "form": ["~", {"path": "name", "value": "Spock"}, ".*"]
    },
    "a130f715-c929-448b-82cd-fe21d3f83b58": {
      "value": true,
      "form": [">=", {"path": ["fact", "spunk"], "value": "10"}, "5"]
    },
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "value": true,
      "form": ["and",
        {

```

```

        "value": true,
        "form": [ ">=", { "path": ["fact", "eyebrow pitch"], "value":
"30"}, "25"]
      },
      {
        "value": true,
        "form": [ "=", { "path": ["fact", "ear-tips"], "value":
"pointed"}, "pointed"]
      },
      {
        "value": true,
        "form": [ "=", { "path": ["fact", "hair"], "value": "dark"},
"dark"]
      },
      {
        "value": true,
        "form": [ ">=", { "path": ["fact", "resting bpm"], "value":
"200"}, "100"]
      },
      {
        "value": true,
        "form": [ "=",
          {
            "path": ["fact", "blood oxygen transporter"],
            "value": "hemocyanin"
          },
          "hemocyanin"
        ]
      }
    ]
  },
  "leaf_groups": {
    "a130f715-c929-448b-82cd-fe21d3f83b58": {
      "name": "Humans",
      "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
      "parent": "00000000-0000-4000-8000-000000000000",
      "rule": [ ">=", ["fact", "spunk"], "5"],
      "environment": "alpha-quadrant",
      "variables": {},
      "classes": {
        "emotion": { "importance": "primary" },
        "logic": { "importance": "secondary" }
      }
    },
    "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
      "name": "Vulcans",
      "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
      "parent": "00000000-0000-4000-8000-000000000000",
      "rule": [ "and", [ ">=", ["fact", "eyebrow pitch"],
"25"],
        [ "=", ["fact", "ear-tips"], "pointed"],
        [ "=", ["fact", "hair"], "dark"],
        [ ">=", ["fact", "resting bpm"], "100"],
        [ "=", ["fact", "blood oxygen
transporter"], "hemocyanin"]
      ],
      "environment": "alpha-quadrant",
      "variables": {},
      "classes": {
        "emotion": { "importance": "ignored" },
        "logic": { "importance": "primary" }
      }
    }
  }
}

```

```

},
"inherited_classifications": {
  "a130f715-c929-448b-82cd-fe21d3f83b58": {
    "environment": "alpha-quadrant",
    "variables": {},
    "classes": {
      "logic": {"importance": "secondary"},
      "emotion": {"importance": "primary"}
    }
  },
  "8aeeb640-8dca-4b99-9c40-3b75de6579c2": {
    "environment": "alpha-quadrant",
    "variables": {},
    "classes": {
      "logic": {"importance": "primary"},
      "emotion": {"importance": "ignored"}
    }
  }
},
"conflicts": {
  "classes": {
    "logic": {
      "importance": [
        {
          "value": "secondary",
          "from": {
            "name": "Humans",
            "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
            ...
          },
          "defined_by": {
            "name": "Humans",
            "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
            ...
          }
        },
        {
          "value": "primary",
          "from": {
            "name": "Vulcans",
            "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
            ...
          },
          "defined_by": {
            "name": "Vulcans",
            "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
            ...
          }
        }
      ]
    },
    "emotion": {
      "importance": [
        {
          "value": "ignored",
          "from": {
            "name": "Vulcans",
            "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
            ...
          },
          "defined_by": {
            "name": "Vulcans",
            "id": "8aeeb640-8dca-4b99-9c40-3b75de6579c2",
            ...
          }
        }
      ]
    }
  }
}

```

```

    }
  },
  {
    "value": "primary",
    "from": {
      "name": "Humans",
      "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
      ...
    },
    "defined_by": {
      "name": "Humans",
      "id": "a130f715-c929-448b-82cd-fe21d3f83b58",
      ...
    }
  }
]
}
},
"individual_classification": {
  "classes": {
    "emotion": {
      "importance": "secondary"
    }
  },
  "variables": {
    "full_name": "S'chn T'gai Spock"
  }
}
}
}

```

Commands endpoint

Use the commands endpoint to unpin specified nodes from all groups they're pinned to.

POST /v1/commands/unpin-from-all

Use the /v1/commands/unpin-from-all to unpin specified nodes from all groups they're pinned to. Nodes that are dynamically classified using rules aren't affected.

Nodes are unpinned from only those groups for which you have view and edit permissions. Because group permissions are applied hierarchically, you must have **Create, edit, and delete child groups** or **Edit child group rules** permissions for the parent groups of the groups you want to unpin the node from.

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

Request format

The request body must be a JSON object describing the nodes to unpin, using the following key:

Key	Definition
nodes	The certname of the nodes (required).

For example:

```
{ "nodes": [ "foo", "bar" ] }
```

Response format

If unpinning is successful, the service returns a list of nodes with the groups they were unpinned from. If a node wasn't pinned to any groups, it's not included in the response.

```
{ "nodes": [ { "name": "foo",
  "groups": [ { "id": "8310b045-c244-4008-88d0-b49573c84d2d",
    "name": "Webservers",
    "environment": "production" },
    { "id": "84b19b51-6db5-4897-9409-a4a3a94b7f09",
    "name": "Test",
    "environment": "test" } ] },
  { "name": "bar",
    "groups": [ { "id": "84b19b51-6db5-4897-9409-a4a3a94b7f09",
      "name": "Test",
      "environment": "test" } ] } ] }
```

Assuming token is set as an environment variable, the example below unpins "host1.example" and "host2.example":

```
type_header= 'Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/commands/unpin-from-all"
data='{ "nodes": [ "host1.example", "host2.example" ] }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The classifier responds with information about each group it is removed from:

```
{ "nodes":
  [ { "name": "host1.example", "groups": [ { "id": "2d83d860-19b4-4f7b-8b70-e5ee4d8646db",
    "name": "test", "environment": "production" } ] },
    { "name": "host2.example", "groups": [ { "id": "2d83d860-19b4-4f7b-8b70-e5ee4d8646db",
    "name": "test", "environment": "production" } ] } ] }
```

Environments endpoint

Use the `environments` endpoint to retrieve information about environments in the node classifier. The output tells you either which environments are available or whether a named environment exists. The output can be helpful when creating new node groups, which must be associated with an environment. The node classifier gets its information about environments from Puppet, so do not use this endpoint to create, update, or delete them.

GET /v1/environments

Use the `/v1/environments` endpoint to retrieve a list of all environments known to the node classifier.

Response format

The response is a JSON array of environment objects, using the following keys:

Key	Definition
<code>name</code>	The name of the environment (a string).
<code>sync_succeeded</code>	Whether the environment synced successfully during the last class synchronization (a Boolean).

Error responses

No error responses specific to this request are expected.

GET /v1/environments/<name>

Use the `/v1/environments/<name>` endpoint to retrieve the environment with the given name. The main use of this endpoint is to check if an environment actually exists.

Response format

If the environment exists, the service returns a 200 response with an environment object in JSON format.

Error responses

If the environment with the given name cannot be found, the service returns a 404: Not Found response with an empty body.

PUT /v1/environments/<name>

Use the `/v1/environments/<name>` endpoint to create a new environment with the given name.

Request format

No further information is required in the request besides the name portion of the URL.

Response format

If the environment is created successfully, the service returns a 201: Created response whose body is the environment object in JSON format.

Error responses

No error responses specific to this operation are expected.

Nodes check-in history endpoint

Use the `nodes` endpoint to retrieve historical information about nodes that have checked into the node classifier.

Enable check-in storage to use this endpoint

Node check-in storage is disabled by default because it can place excessive loads on larger deployments. You must enable node check-in storage before using the check-in history endpoint. If node check-in storage is not enabled, the endpoint returns an empty array.

To enable node check-in storage, set the `classifier_node_check_in_storage` parameter in the `puppet_enterprise::profile::console` class to `true`.

Related information

[Set configuration data](#) on page 323

Configuration data set in the console is used for automatic parameter lookup, the same way that Hieradata is used. Console configuration data takes precedence over Hieradata, but you can combine data from both sources to configure nodes.

GET /v1/nodes

Use the `/v1/nodes` endpoint to retrieve a list of all nodes that have checked in with the node classifier, each with their check-in history.

Query Parameters**limit**

Controls the maximum number of nodes returned. `limit=10` returns 10 nodes.

offset

Specifies how many nodes to skip before the first returned node. `offset=20` skips the first 20 nodes.



CAUTION: In deployments with large numbers of nodes, a large or unspecified `limit` might cause the console-services process to run out of memory and crash.

Response format

The response is a JSON array of node objects. Each node object contains these two keys:

Key	Definition
<code>name</code>	The name of the node according to Puppet (a string).
<code>check_ins</code>	An array of check-in objects (described below).

Each check-in object describes a single check-in of the node. The check-in objects have the following keys:

Key	Definition
<code>time</code>	The time of the check-in as a string in ISO 8601 format (with timezone).
<code>explanation</code>	An object mapping between IDs of groups that the node was classified into and explained condition objects that describe why the node matched this group's rule.
<code>transaction_uuid</code>	A uuid representing a particular Puppet transaction that is submitted by Puppet at classification time. This makes it possible to identify the check-in involved in generating a specific catalog and report.

The explained condition objects are the node group's rule condition marked up with the node's value and the result of evaluation. Each form in the rule (that is, each array in the JSON representation of the rule condition) is replaced with an object that has two keys:

Key	Definition
<code>value</code>	A Boolean that is the result of evaluating this form. At the top level, this is the result of the entire rule condition, but because each sub-condition is marked up with its value, you can use this to understand, say, which parts of an <code>or</code> condition were true.
<code>form</code>	The condition form, with all sub-forms as further explained condition objects.

Besides the condition markup, the comparison operations of the rule condition have their first argument (the fact path) replaced with an object that has both the fact path and the value that was found in the node at that path.

The following example shows the format of an explained condition.

Start with a node group with the following rule:

```
[ "and", [ ">=", [ "fact", "pressure hulls" ], "1" ],
          [ "=", [ "fact", "warp cores" ], "0" ],
          [ ">=", [ "fact", "docking ports" ], "10" ] ]
```

The following node checks into the classifier:

```
{
```



```

    "name": "Deep Space 9",
    "fact": {
      "pressure hulls": "10",
      "docking ports": "18",
      "docking pylons": "3",
      "warp cores": "0",
      "bars": "1"
    }
  }
}

```

When the node checks in for classification, it matches the above rule, so that check-in's explanation object has an entry for the node group that the rule came from. The value of this entry is this explained condition object:

```

{
  "value": true,
  "form": [
    "and",
    {
      "value": true,
      "form": [">=", {"path": ["fact", "pressure hulls"], "value": "3"}],
      "1"
    },
    {
      "value": true,
      "form": ["=", {"path": ["fact", "warp cores"], "value": "0"}], "0"
    },
    {
      "value": true,
      "form": [">", {"path": ["fact", "docking ports"], "value": "18"}], "9"
    }
  ]
}

```

GET /v1/nodes/<node>

Use the /v1/nodes/<node> endpoint to retrieve the check-in history for only the specified node.

Response format

The response is one node object as described above in the GET /v1/nodes documentation, for the specified node. The following example shows a node object:

```

{
  "name": "Deep Space 9",
  "check_ins": [
    {
      "time": "2369-01-04T03:00:00Z",
      "explanation": {
        "53029cf7-2070-4539-87f5-9fc754a0f041": {
          "value": true,
          "form": [
            "and",
            {
              "value": true,
              "form": [">=", {"path": ["fact", "pressure hulls"], "value":
"3"}], "1"
            },
            {
              "value": true,
              "form": ["=", {"path": ["fact", "warp cores"], "value": "0"}],
"0"
            }
          ]
        }
      }
    }
  ]
}

```

```

    {
      "value": true,
      "form": [ ">" { "path": [ "fact", "docking ports"], "value":
"18"}, "9"]
    }
  ]
}
],
"transaction_uuid": "d3653a4a-4ebe-426e-a04d-dbebec00e97f"
}

```

Error responses

If the specified node has not checked in, the service returns a 404: Not Found response, with the usual JSON error response in its body.

Group children endpoint

Use the group children endpoint to retrieve a specified group and its descendents.

GET /v1/group-children/:id

Use the /v1/group-children/:id endpoint to retrieve a specified group and its descendents.

Request format

The request body must be a JSON object specifying a group and an optional depth indicating how many levels of descendents to return.

- **depth:** (optional) an integer greater than or equal to 0 that limits the depth of trees returned. Zero means return the group with no children.

For example:

```
GET /v1/group-children/00000000-0000-4000-8000-000000000000?depth=2
```

Response format

The response is a JSON array of group objects, using the following keys:

Key	Definition
name	The name of the node group (a string).
id	The node group's ID, which is a string containing a type-4 (random) UUID.
description	An optional key containing an arbitrary string describing the node group.
environment	The name of the node group's environment (a string), which indirectly defines what classes are available for the node group to set, and is the environment that nodes classified into this node group run under.

Key	Definition
<code>environment_trumps</code>	This is a boolean that changes the behavior of classifications that this node group is involved in. The default behavior is to return a classification-conflict error if the node groups that a node is classified into do not all have the same environment. If this flag is set, then this node group's environment overrides the environments of the other node groups (provided that none of them also have this flag set), with no attempt made to validate that the other node groups' classes and class parameters exist in this node group's environment.
<code>parent</code>	The ID of the node group's parent (a string). A node group is not permitted to be its own parent, unless it is the All Nodes group (which is the root of the hierarchy). Note that the root group always has the lowest-possible random UUID, 00000000-0000-4000-8000-000000000000.
<code>rule</code>	A boolean condition on node properties. When a node's properties satisfy this condition, it's classified into the node group. See Rule condition grammar for more information on how this condition must be structured.
<code>classes</code>	An object that defines both the classes consumed by nodes in this node group and any non-default values for their parameters. The keys of the object are the class names, and the values are objects describing the parameters. The parameter objects' keys are parameter names, and the values are what the node group sets for that parameter (always a string).
<code>deleted</code>	An object similar the <code>classes</code> object that shows which classes and class parameters set by the node group have since been deleted from Puppet. If none of the node group's classes or parameters have been deleted, this key is not present, so checking the presence of this key is an easy way to check whether the node group has references that need updating. The keys of this object are class names, and the values are further objects. These secondary objects always contain the <code>puppetlabs.classifier/deleted</code> key, whose value is a Boolean indicating whether the entire class has been deleted from Puppet. The other keys of these objects are parameter names, and the other values are objects that always contain two keys: <code>puppetlabs.classifier/deleted</code> , mapping to a boolean indicating whether the specific class parameter has been deleted from Puppet; and <code>value</code> , mapping to the string value set by the node group for this parameter (the value is duplicated for convenience, as it also appears in the <code>classes</code> object).
<code>variables</code>	An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).

Key	Definition
children	A JSON array of the group's immediate children. Children of children are included to the optionally-specified depth.
immediate_child_count	The number of immediate children of the group. Child count reflects the number of children that exist in the classifier, not the number that are returned in the request, which can vary based on permissions and query parameters.

The following is an example response from a query of the root node group with two children, each with three children. The user has permission to view only `child-1` and `grandchild-5`, which limits the response.

```
[
  {
    "name": "child-1",
    "id": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
    "parent": "00000000-0000-4000-8000-000000000000",
    "environment_trumps": false,
    "rule": ["and", ["=", ["fact", "foo"], "bar"], ["not", ["<",
["fact", "uptime_days"], "31"]]],
    "variables": {},
    "environment": "test",
    "classes": {},
    "children": [
      {
        "name": "grandchild-1",
        "id": "a3d976ad-51d3-4a29-af57-09990f3a2481",
        "parent": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
        "environment_trumps": false,
        "rule": ["and", ["=", ["fact", "foo"], "bar"], ["or", ["~",
"name", "db"], ["<", ["fact", "processorcount"], "9"], ["=", ["fact",
"operatingsystem"], "Ubuntu"]]],
        "variables": {},
        "environment": "test",
        "classes": {},
        "children": [],
        "immediate_child_count": 0
      },
      {
        "name": "grandchild-2",
        "id": "71905c11-5295-41cf-a143-31b278cfc859",
        "parent": "652227cd-af24-4fd8-96d4-b9b55ca28efb",
        "environment_trumps": false,
        "rule": ["and", ["=", ["fact", "foo"], "bar"], ["not", ["~",
["fact", "kernel"], "SunOS"]]],
        "variables": {},
        "environment": "test",
        "classes": {},
        "children": [],
        "immediate_child_count": 0
      }
    ],
    "immediate_child_count": 2
  },
  {
    "name": "grandchild-5",
    "id": "0bb94f26-2955-4adc-8460-f5ce244d5118",
    "parent": "0960f75e-cdd0-4966-96f6-5e60948a7217",
    "environment_trumps": false,
```

```

    "rule": ["and", ["=", ["fact", "foo"], "bar"], ["and", ["<",
["fact", "processorcount"], "16"], [">=", ["fact", "kernelmajversion"],
"2"]]]],
    "variables": {},
    "environment": "test",
    "classes": {},
    "children": [],
    "immediate_child_count": 0
  }
]

```

Permissions

The response returned varies based on your permissions.

Permissions	Response
View the specified group only	An array containing the group and its descendents, ending at the optional depth
View descendents of the specified group, but not the group itself	An array starting at the roots of every tree you have permission to view and ending at the optional depth
View neither the specified group nor its descendents	An empty array

Error responses

Responses and keys returned for group requests depend on the type of error.

malformed-uuid

If the requested ID is not a valid UUID, the service returns a 400: Bad Request response using the following keys:

Key	Definition
kind	"malformed-uuid"
details	The malformed UUID as received by the server.

malformed-number or illegal-count

If the value of the depth parameter is not an integer, or is a negative integer, the service returns a 400: Bad Request response using one of the following keys:

Key	Definition
kind	"malformed-number" or "illegal-count"

Rules endpoint

Use the rules endpoint to translate a group's rule condition into PuppetDB query syntax.

POST /v1/rules/translate

Translate a group's rule condition into PuppetDB query syntax.

Request format

The request's body contains a rule condition as it would appear in the rule field of a group object.

The endpoint supports an optional query parameter `format`, which defaults to `nodes`. If specified as `?format=inventory`, it allows you to get the classifier rules in a compatible [dot notation](#) format instead of the standard [PuppetDB AST](#).

Response format

The response is a PuppetDB query string that can be used with PuppetDB nodes endpoint in order to see which nodes would satisfy the rule condition (that is, which nodes would be classified into a group with that rule).

Error responses

Rules that use structured or trusted facts cannot be converted into PuppetDB queries, because PuppetDB does not yet support structured or trusted facts. If the rule cannot be translated into a PuppetDB query, the server returns a 422 Unprocessable Entity response containing the usual JSON error object. The error object has a `kind` of "untranslatable-rule", a `msg` that describes why the rule cannot be translated, and contains the received rule in `details`.

If the request does not contain a valid rule, the server returns a 400 Bad Request response with the usual JSON error object. If the rule was not valid JSON, the error's `kind` is "malformed-request", the `msg` states that the request's body could not be parsed as JSON, and the `details` contain the request's body as received by the server.

If the rule does not conform to the rule grammar, the `kind` key is "schema-violation", and the `details` key is an object with `submitted`, `schema`, and `error` keys which respectively describe the submitted object, the schema that object is expected to conform to, and how the submitted object failed to conform to the schema.

Related information

[GET /v1/groups](#) on page 389

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

Import hierarchy endpoint

Use the import hierarchy endpoint to delete all existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.

POST /v1/import-hierarchy

Delete *all* existing node groups from the node classifier service and replace them with the node groups in the body of the submitted request.

The request's body must contain an array of node groups that form a valid and complete node group hierarchy. Valid means that the hierarchy does not contain any cycles, and complete means that every node group in the hierarchy is reachable from the root.

Request format

The request body must be a JSON array of node group objects as described in the `groups` endpoint documentation. All fields of the node group objects must be defined; no default values are supplied by the service.

Note that the output of the group collection endpoint, `/v1/groups`, is valid input for this endpoint.

Response format

If the submitted node groups form a complete and valid hierarchy, and the replacement operation is successful, a 204 No Content response with an empty body is returned.

Error responses

If any of the node groups in the array are malformed, a 400 Bad Request response is returned. The response contains the usual JSON error payload. The `kind` key is "schema-violation"; the `msg` key contains a short description of the problems with the malformed node groups; and the `details` key contains an object with three keys:

- `submitted`: an array of only the malformed node groups found in the submitted request.

- **error**: an array of structured descriptions of how the node group at the corresponding index in the submitted array failed to meet the schema.
- **schema**: the structured schema for node group objects.

If the hierarchy formed by the node groups contains a cycle, then a 422 Unprocessable Entity response is returned. The response contains the usual JSON error payload, where the **kind** key is "inheritance-cycle", the **msg** key contains the names of the node groups in the cycle, and the **details** key contains an array of the complete node group objects in the cycle.

If the hierarchy formed by the node groups contains node groups that are unreachable from the root, then a 422 Unprocessable Entity response is returned. The response contains the usual JSON error payload, where the **kind** key is "unreachable-groups", the **msg** lists the names of the unreachable node groups, and the **details** key contains an array of the unreachable node group objects.

Related information

[Groups endpoint](#) on page 389

The [groups](#) endpoint is used to create, read, update, and delete groups.

[Node classifier errors](#) on page 435

Familiarize yourself with error responses to make working the node classifier service API easier.

Last class update endpoint

Use the last class update endpoint to retrieve the time that classes were last updated from the primary server.

GET /v1/last-class-update

Use the `/v1/last-class-update` endpoint to retrieve the time that classes were last updated from the primary server.

Response

The response is always an object with one field, `last_update`. If there has been an update, the value of `last_update` field is the time of the last update in ISO8601 format. If the node classifier has never updated from Puppet, the field is null.

Update classes endpoint

Use update classes endpoint to trigger the node classifier to update class and environment definitions from the primary server.

POST /v1/update-classes

Use the `/v1/update-classes` endpoint to trigger the node classifier to update class and environment definitions from the primary server. The classifier service uses this endpoint when you refresh classes in the console.

Note: If you don't use Code Manager *and* you changed the default value of the `environment-class-cache-enabled` server setting, you must [manually delete the environment cache](#) before using this endpoint.

Query parameters

The request accepts the following optional parameter:

Parameter	Value
<code>environment</code>	If provided, fetches classes for only the specified environment.

For example:

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
```

```
uri="https://$(puppet config print server):4433/classifier-api/v1/update-classes?environment=production"

curl --cert "$cert" --cacert "$cacert" --key "$key" --request POST "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response

For a successful update, the service returns a 201 response with an empty body.

Error responses

If the primary server returns an unexpected status to the node classifier, the service returns a 500: `Server Error` response with the following keys:

Key	Definition
kind	"unexpected-response"
msg	Describes the error
details	A JSON object, which has <code>url</code> , <code>status</code> , <code>headers</code> , and <code>body</code> keys describing the response the classifier received from the primary server

Related information

[Enable or disable cached data when updating classes](#) on page 163

The optional `environment-class-cache-enabled` setting specifies whether cached data is used when updating classes in the console. When `true`, Puppet Server refreshes classes using file sync, improving performance.

Validation endpoints

Use validation endpoints to validate groups in the node classifier.

POST /v1/validate/group

Use the `/v1/validate/group` endpoint to validate groups in the node classifier.

Request format

The request contains a group object. The request uses the following keys:

Key	Definition
name	The name of the node group (required).
environment	The name of the node group's environment. This key is optional; if it's not provided, the default environment (<code>production</code>) is used.
environment_trumps	Whether this node group's environment overrides those of other node groups at classification-time. This key is optional; if it's not provided, the default value of <code>false</code> is used.
description	A string describing the node group. This key is optional; if it's not provided, the node group has no description and this key doesn't appear in responses.
parent	The ID of the node group's parent (required).

Key	Definition
<code>rule</code>	The condition that must be satisfied for a node to be classified into this node group. The structure of this condition is described in the "Rule Condition Grammar" section above.
<code>variables</code>	An object that defines the names and values of any top-level variables set by the node group. The keys of the object are the variable names, and the corresponding value is that variable's value, which can be any sort of JSON value. The <code>variables</code> key is optional, and if a node group does not define any top-level variables then it can be omitted.
<code>classes</code>	An object that defines the classes to be used by nodes in the node group. The <code>classes</code> key is required, and at minimum is an empty object (<code>{}</code>). The <code>classes</code> key also contains the parameters for each class. Some classes have required parameters. This is a two-level object; that is, the keys of the object are class names (strings), and each key's value is another object that defines class parameter values. This innermost object maps between class parameter names and their values. The keys are the parameter names (strings), and each value is the parameter's value, which can be any kind of JSON value. The <code>classes</code> key is not optional; if it is missing, the service returns a 400: Bad Request response.

Response format

If the group is valid, the service returns a 200 OK response with the validated group as the body.

If a validation error is encountered, the service returns one of the following 400-level error responses.

Responses and keys returned for create group requests depend on the type of error.

`schema-violation`

If any of the required keys are missing or the values of any of the defined keys do not match the required type, the service returns a 400: Bad Request response using the following keys:

Key	Definition
<code>kind</code>	"schema-violation"
<code>details</code>	<p>An object that contains three keys:</p> <ul style="list-style-type: none"> <code>code>submitted</code>: Describes the submitted object.</code> <code>schema</code>: Describes the schema that object is expected to conform to. <code>error</code>: Describes how the submitted object failed to conform to the schema.

`malformed-request`

If the request's body could not be parsed as JSON, the service returns a `400: Bad Request` response using the following keys:

Key	Definition
<code>kind</code>	"malformed-request"
<code>details</code>	An object that contains two keys: <ul style="list-style-type: none"> <code>body</code>: Holds the request body that was received. <code>error</code>: Describes how the submitted object failed to conform to the schema.

`uniqueness-violation`

If your attempt to create the node group violates uniqueness constraints (such as the constraint that each node group name must be unique within its environment), the service returns a `422: Unprocessable Entity` response using the following keys:

Key	Definition
<code>kind</code>	"uniqueness-violation"
<code>msg</code>	Describes which fields of the node group caused the constraint to be violated, along with their values.
<code>details</code>	An object that contains two keys: <ul style="list-style-type: none"> <code>conflict</code>: An object whose keys are the fields of the node group that violated the constraint and whose values are the corresponding field values. <code>constraintName</code>: The name of the database constraint that was violated.

`missing-referents`

If classes or class parameters defined by the node group, or inherited by the node group from its parent, do not exist in the submitted node group's environment, the service returns a `422: Unprocessable Entity` response. In both cases the response object uses the following keys:

Key	Definition
<code>kind</code>	"missing-referents"
<code>msg</code>	Describes the error and lists the missing classes or parameters.

Key	Definition
details	<p>An array of objects, where each object describes a single missing referent, and has the following keys:</p> <ul style="list-style-type: none"> • <code>kind</code>: "missing-class" or "missing-parameter", depending on whether the entire class doesn't exist, or the class just doesn't have the parameter. • <code>missing</code>: The name of the missing class or class parameter. • <code>environment</code>: The environment that the class or parameter is missing from; that is, the environment of the node group where the error was encountered. • <code>group</code>: The name of the node group where the error was encountered. Due to inheritance, this might not be the group where the parameter was defined. • <code>defined_by</code>: The name of the node group that defines the class or parameter.

missing-parent

If the parent of the node group does not exist, the service returns a `422: Unprocessable Entity` response. The response object uses the following keys:

Key	Definition
kind	"missing-parent"
msg	Shows the parent UUID that did not exist.
details	The full submitted node group.

inheritance-cycle

If the request causes an inheritance cycle, the service returns a `422: Unprocessable Entity` response. The response object uses the following keys:

Key	Definition
kind	"inheritance-cycle"
details	An array of node group objects that includes each node group involved in the cycle
msg	A shortened description of the cycle, including a list of the node group names with each followed by its parent until the first node group is repeated.

Node classifier errors

Familiarize yourself with error responses to make working the node classifier service API easier.

Error response description

Errors from the node classifier service are JSON responses.

Error responses contain these keys:

Key	Definition
<code>kind</code>	A string classifying the error. It is the same for all errors that have the same kind of thing in their <code>details</code> key.
<code>msg</code>	A human-readable message describing the error, suitable for presentation to the user.
<code>details</code>	Additional machine-readable information about the error condition. The format of this key's value varies between kinds of errors but is the same for any given error kind.

Internal server errors

Any endpoint might return a `500: Internal Server Error` response in addition to its usual responses. There are two kinds of internal server error responses: `application-error` and `database-corruption`.

An `application-error` response is a catchall for unexpected errors. The `msg` of an `application-error` 500 contains the underlying error's message first, followed by a description of other information that can be found in `details`. The `details` contain the error's stack trace as an array of strings under the `trace` key, and might also contain `schema`, `value`, and `error` keys if the error was caused by a schema validation failure.

A `database-corruption` 500 response occurs when a resource that is retrieved from the database fails to conform to the schema expected of it by the application. This is probably just a bug in the software, but it could potentially indicate either genuine corruption in the database or that a third party has changed values directly in the database. The `msg` section contains a description of how the database corruption could have occurred. The `details` section contains `retrieved`, `schema`, and `error` keys, which have the resource as retrieved, the schema it is expected to conform to, and a description of how it fails to conform to that schema as the respective values.

Not found errors

Any endpoint where a resource identifier is supplied can produce a `404 Not Found Error` response if a resource with that identifier could not be found.

All not found error responses have the same form. The `kind` is "not-found", the `msg` is "The resource could not be found.", and the `details` key contains the URI of the request that resulted in this response.

Node classifier API v2

These are the endpoints for the node classifier v2 API.

- [Classification endpoint](#) on page 437

The `classification` endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

Related information

[API index](#) on page 26

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

Classification endpoint

The `classification` endpoint takes a node name and a set of facts, and returns information about how that node is classified. The output can help you test your classification rules.

- [POST /v2/classified/nodes/<name>](#) on page 437

Use the `/v2/classified/nodes/\<name\>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

POST /v2/classified/nodes/<name>

Use the `/v2/classified/nodes/\<name\>` endpoint to retrieve the classification information for the node with the given name and facts as supplied in the body of the request.

Request format

The request body can contain a JSON object describing the facts and trusted facts of the node to be classified. The object can have these keys:

Key	Definition
<code>fact</code>	The regular, non-trusted facts of the node. The value of this key is a further object, whose keys are fact names, and whose values are the fact values. Fact values can be a string, number, boolean, array, or object.
<code>trusted</code>	The trusted facts of the node. The values of this key are subject to the same restrictions as those on the value of the <code>fact</code> key.

Response format

The response is a JSON object describing the node post-classification, using these keys:

Key	Definition
<code>name</code>	The name of the node (a string).
<code>groups</code>	An array of the groups that this node was classified into. The value of this key is an array of hashes containing the <code>id</code> and the <code>name</code> , sorted by name
<code>environment</code>	The name of the environment that this node uses, which is taken from the node groups the node was classified into.
<code>classes</code>	An array of the classes (strings) that this node received from the groups it was classified into.
<code>parameters</code>	An object describing class parameter values for the above classes wherever they differ from the default. The keys of this object are class names, and the values are further objects describing the parameters for just the associated class. The keys of that innermost object are parameter names, and the values are the parameter values, which can be any sort of JSON value.

This is an example of a response from this endpoint:

```
{
  "name": "foo.example.com",
  "groups": [{ "id": "9c0c7d07-a199-48b7-9999-3cdf7654e0bf",
    "name": "a group" },
    { "id": "96d1a058-225d-48e2-a1a8-80819d31751d",
    "name": "b group" } ],
  "environment": "staging",
  "classes": ["apache"],
  "parameters": {
    "apache": {
      "keepalive_timeout": 30,
      "log_level": "notice"
    }
  }
}
```

Error responses

If the node is classified into multiple node groups that define conflicting classifications for the node, the service returns a 500 Server Error response.

The body of this response contains the usual JSON error object described in the errors documentation.

The `kind` key of the error is "classification-conflict", the `msg` describes generally why this happens, and the `details` key contains an object that describes the specific conflicts encountered.

The keys of these objects are the names of parameters that had conflicting values defined, and the values are arrays of value detail objects. The details object may have between one and all of the following three keys.

Key	Definition
environment	Maps directly to an array of value detail objects (described below).
variables	Contains an object with a key for each conflicting variable, whose values are an array of value detail objects.
classes	Contains an object with a key for each class that had conflicting parameter definitions, whose values are further objects that describe the conflicts for that class's parameters.

A value details object describes one of the conflicting values defined for the environment, a variable, or a class parameter. Each object contains these keys:

Key	Definition
value	The defined value, which is a string for environment and class parameters, but for a variable can be any JSON value.
from	The node group that the node was classified into that caused this value to be added to the node's classification. This group cannot define the value, because it can be inherited from an ancestor of this group.
defined_by	The node group that actually defined this value. This is often the <code>from</code> group, but could instead be an ancestor of that group.

This example shows a classification conflict error object with node groups truncated for clarity:

```
{
  "kind": "classification-conflict",
  "msg": "The node was classified into multiple unrelated groups that
defined conflicting class parameters or top-level variables. See `details`
for a list of the specific conflicts.",
  "details": {
    "classes": {
      "songColors": {
        "blue": [
          {
            "value": "Blue Suede Shoes",
            "from": {
              "name": "Elvis Presley",
              "classes": {},
              "rule": ["=", "nodename", "the-node"],
              ...
            },
            "defined_by": {
              "name": "Carl Perkins",
              "classes": {"songColors": {"blue": "Blue Suede Shoes"}},
              "rule": ["not", ["=", "nodename", "the-node"]],
              ...
            }
          },
          {
            "value": "Since You've Been Gone",
            "from": {
              "name": "Aretha Franklin",
              "classes": {"songColors": {"blue": "Since You've Been Gone"}},
              ...
            },
            "defined_by": {
              "name": "Aretha Franklin",
              "classes": {"songColors": {"blue": "Since You've Been Gone"}},
              ...
            }
          }
        ]
      }
    }
  }
}
```

In this example, the conflicting "Blue Suede Shoes" value was included in the classification because the node matched the "Elvis Presley" group (since that is the value of the "from" key), but that group doesn't define the "Blue Suede Shoes" value. That value is defined by the "Carl Perkins" group, which is an ancestor of the "Elvis Presley" group, causing the latter to inherit the value from the former. The other conflicting value, "Since You've Been Gone", is defined by the same group that the node matched.

Node inventory API

These are the endpoints for the node inventory v1 API.

Related information

[API index](#) on page 26

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

Node inventory API: forming requests

Make well-formed HTTP(S) requests to the Puppet inventory service API.

By default, the node inventory service listens on port 8143, and all endpoints are relative to the `/inventory/v1` path. For example, the full URL for the `/command/create-connection` endpoint on localhost is `https://localhost:8143/inventory/v1/command/create-connection`.

Token authentication

All requests made to the inventory service API require authentication. You do this for each endpoint in the service using user authentication tokens contained within the `X-Authentication` request header.

Example token usage: create a connection entry

To post a new connection entry to the inventory service when running on localhost, first generate a token with the `puppet-access` tool.

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/command/create-connection"
data='{ "certnames": [ "new.node" ],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      },
      "sensitive_parameters": {
        "username": "root",
        "password": "password"
      },
      "duplicates": "replace"
    }'
```

```
curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Example token usage: query connections for a certname

To query the `/query/connections` endpoint, use the same token and header pattern.

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/query/connections?certname='new.node'"

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri"
```

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

POST /command/create-connection

Create a new connection entry in the inventory service database.

Connection entries contain connection options, such as credentials, which are used to connect to the certnames provided in the payload via the provided connection type.

This endpoint also inserts each of the provided certnames into PuppetDB with an empty fact set, if they are not already present. After certnames have been added to PuppetDB, you can view them from the **Nodes** page in the Puppet Enterprise console. You can also add them to an inventory node list when you set up a job to run tasks.

Important: When the Puppet orchestrator targets a certname to run a task, it first considers the value of the `hostname` key present in the `parameters`, if available. Otherwise, it uses the value of the `certnames` key as the hostname. A good practice is to include a `hostname` key only when the hostname differs from the `certname`. Do not include a `hostname` key for multiple `certname` connection entries.

Request format

The request body must be a JSON object.

certnames

required Array containing the list of certnames to associate with connection information.

type

required String containing either `ssh` or `winrm`. Instructs bolt-server which connection type to use to access the node when running a task.

parameters

required Object containing arbitrary key/value pairs. The necessary parameters for connecting to the provided certnames.



CAUTION: A `hostname` key entered here takes precedence over the values in `certnames` key.

sensitive_parameters

required Object containing arbitrary key/value pairs. The necessary sensitive data for connecting to the provided certnames, stored in an encrypted format.

duplicates

required String containing either `error` or `replace`. Instructs how to handle cases where one or more provided certnames conflict with existing certnames stored in the inventory connections database. `error` results in a 409 response if any certnames are duplicates. `replace` overwrites the existing certnames if there are conflicts.

Request examples

```
{
  "certnames": ["sshnode1.example.com", "sshnode2.example.com"],
  "type": "ssh",
  "parameters": {
    "port": 1234,
    "connect-timeout": 90,
    "user": "inknowahorse",
    "run-as": "fred"
  },
  "sensitive_parameters": {
    "password": "password",
    "sudo-password": "xtheowl"
  }
}
```

```

    },
    "duplicates": "replace"
  }
}

```

```

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/command/create-connection"
data='{
  "certnames": [
    "sshnode1.example.com",
    "sshnode2.example.com"
  ],
  "type": "ssh",
  "parameters": {
    "port": 1234,
    "connect-timeout": 90,
    "user": "inknowahorse",
    "run-as": "fred"
  },
  "sensitive_parameters": {
    "password": "password",
    "sudo-password": "xtheowl"
  },
  "duplicates": "replace"
}'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"

```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response format

If the request is valid according to the schema and the entry is successfully recorded in the database, the server returns a 201 response. The response has the same format for single and multiple certname entries.

The response is a JSON object containing the `connection_id`.

`connection_id`

A unique identifier that can be used to reference the record.

Response example

```

{
  "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec"
}

```

SSH parameters

To create a connection with type `ssh`, the following parameters and sensitive parameters are valid.

`parameters`

Required. Object containing the following key/value pairs and the necessary parameters for connecting to the provided certnames.

`user`

Required string. The user to log in as when connecting to the host.

`port`

Optional integer. Connection port with the default of 22.

connect-timeout

Optional integer. The length of time, in seconds, PE should wait when establishing connections.

run-as

Optional string. After login, the user name to use for running commands. The default is the same value as `user`.

tmpdir

Optional string. The directory to upload and execute temporary files on the target, if different than `/temp`.

tty

Optional boolean. Enable text terminal allocation.

hostname

Optional string. The hostname to connect to. Should only be provided if the desired `certname` for the newly created node within PE is different than the `hostname` of the intended target, in which case the `certname` field in the request body should be the desired `certname`, and the `hostname` field in the parameters object should be the `hostname` to connect to. As a good practice, do not provide a `hostname` key for connection entries with multiple `certnames`, and only include it if the `certname` is different from the `hostname` of the intended target. If left unspecified, the Orchestrator will connect to each host using its `certname` as the `hostname`.

sensitive_parameters

Required. Object containing arbitrary key/value pairs and the necessary sensitive data for connecting to the provided `certnames`, to be stored in encrypted format.

password

Optional string. Password to authenticate. One of either `password` or `private-key-content` is required.

private-key-content

Optional string. Contents of a private key. One of either `password` or `private-key-content` is required.

sudo-password

Optional string. Password to use when changing users via `run-as`. Should only be included if `run-as` is specified in the parameters object.

WinRM parameters

For creating a connection with type `winrm`, the following parameters and sensitive parameters are valid.

parameters

Required. Object containing the following key/value pairs and the necessary parameters for connecting to the provided `certnames`.

user

Required string. The user to log in as when connecting to the host.

port

Optional integer. Connection port with the default of 22.

connect-timeout

Optional integer. The length of time, in seconds, PE should wait when establishing connections.

tmpdir

Optional string. The directory to upload and execute temporary files on the target, if different than `/temp`.

extensions

Optional array. List of file extensions that are accepted for tasks.

hostname

Optional string. The hostname to connect to. Should only be provided if the desired certname for the newly created node within PE is different than the hostname of the intended target, in which case the certname field in the request body should be the desired certname, and the hostname field in the parameters object should be the hostname to connect to. As a good practice, do not provide a hostname key for connection entries with multiple certnames, and only include it if the certname is different from the hostname of the intended target. If left unspecified, the Orchestrator will connect to each host using its certname as the hostname.

sensitive_parameters

Required. Object containing arbitrary key/value pairs and the necessary sensitive data for connecting to the provided certnames, to be stored in an encrypted format.

password

Required string. The password used to authenticate.

POST /command/delete-connection

Delete certnames from all associated connection entries in the inventory service database and update those certname entries in the PuppetDB with the flag `preserve: false`.

Request format

The request body must be a JSON object containing a `certnames` key.

certnames

required Array containing the list of certnames to be removed.

Request examples

```
{
  "certnames": ["avery.gooddevice", "amediocre.device"]
}
```

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/inventory/v1/command/delete-connection"
data='{ "certnames": ["avery.gooddevice", "amediocre.device"] }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response format

If the request matches the schema and is processed successfully, the service responds with a 204 status code and no body payload. If the user is unauthorized, the service responds with a 403 error code.

Command endpoint error responses

General format of error responses.

Every error response from the inventory service is a JSON response. Each response is an object the contains the following keys:

kind

The kind of error encountered.

msg

The message associated with the error.

details

A hash with more information about the error.

For example, if the request is missing the required content type headers, the following error occurs:

```
{
  "kind" : "puppetlabs.inventory/not-acceptable",
  "msg" : "accept must include content-type json",
  "details" : ""
}
```

kind error responses

For this endpoint, the `kind` key of the error displays the conflict.

puppetlabs.inventory/unknown-error

If an unknown error occurs during the request, the server returns a 500 response.

puppetlabs.inventory/not-acceptable

If the content provided to the API contains an "accepts" header which does not allow for JSON, the server returns a 406 response.

puppetlabs.inventory/unsupported-type

If the content provided to the API contains a "content-type" header other than JSON, the server returns a 416 response.

puppetlabs.inventory/json-parse-error

If there is an error while processing the request body, the server returns a 400 response.

puppetlabs.inventory/schema-validation-error

If there is a violation of the required format for the request body, the server returns a 400 response.

puppetlabs.inventory/not-permitted

If the user requesting an action does not have the necessary permissions to do so, the server returns a 403 response.

puppetlabs.inventory/duplicate-certnames

If the `duplicates` parameter is not set, or is set as `error`, and one or more of the certnames in the request body already exist within the inventory, the server returns a 409 response.

GET /query/connections

List all the connections entries in the inventory database.

Request format

The request body must be a JSON object.

certname

optional String that represents the single certname to retrieve.

sensitive

optional String or boolean that instructs whether to return sensitive parameters for each connection in the response. This parameter is gated by permission validation.

extract

optional Array of keys to return for each certname. `connection_id` is always returned, regardless of whether it is included. When `extract` is not present in the body, all keys are returned.

Tip: In order to return sensitive parameters in the extract list, the `sensitive` query parameter must be present and resolve to true. Otherwise, they are excluded.

Response format

The response is a JSON object containing the known connections. The following keys are used:

items

Contains an array of all the known connections matching the specified (or not specified) filtering criteria. Each item under `items` is an object with the following keys:

connection_id

String that is the unique identifier for the connections entry.

certnames

Array of strings that contains the certnames of the matching connections entries.

type

String that describes the type of connection for the given information. For example, `ssh` or `winrm`.

parameters

Object containing arbitrary key/value pairs and describes connection options for the entry.

sensitive_parameters

when specified and permitted An object that contains arbitrary key/value pairs and describes the sensitive connection options for the entry.

Response examples

A response for a request to list all the connections entries in the inventory database: `GET /query/connections`

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["devices.arecool", "dont.youthink"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    },
    {
      "connection_id": "4932bfe7-69c4-412f-b15c-ac0a7c2883f1",
      "certnames": ["managing.devices", "is.evencooler"],
      "type": "winrm",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```

A response for a request to list a specific certname: `GET /query/connections?certname="averygood.device"`

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
    }
  ]
}
```

```

    "type": "ssh",
    "parameters": {
      "tmpdir": "/tmp",
      "port": 1234
    }
  ]
}

```

A response for request to list a specific certname and its sensitive parameters: GET /query/connections?certname="averygood.device"&sensitive=true example

```

{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      },
      "sensitive_parameters": {
        "username": "johnjohnjimmyjohn",
        "password": "C7b0$ls8lt0"
      }
    }
  ]
}

```

A response for request to list a specific certname and its value for the type key: GET /query/connections?certname="averygood.device"&extract=["type"]

```

{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh"
    }
  ]
}

```

POST /query/connections

Retrieve inventory connections in bulk via certname.

Request format

The request body must be a JSON object.

certnames

Array containing the list of certnames to retrieve from the inventory database. If this key is not included, all connections are returned.

extract

Array of keys to return for each certname. `connection_id` is always returned, regardless of whether it is included. When `extract` is not present in the body, all keys are returned.

Tip: In order to return sensitive parameters in the extract list, the `sensitive` query parameter must be present and resolve to true. Otherwise, they are excluded.

sensitive

optional String or boolean that instructs whether to return sensitive parameters for each connection in the response. This parameter is gated by permission validation.

Response format

The response is a JSON object containing the known connections. The following keys are used:

items

Contains an array of all the known connections matching the specified (or not specified) filtering criteria. Each item under `items` is an object with the following keys:

connection_id

String that is the unique identifier for the connections entry.

certnames

Array of strings that contains the certnames of the matching connections entries.

type

String that describes the type of connection for the given information. For example, `ssh` or `winrm`.

parameters

Object containing arbitrary key/value pairs and describes connection options for the entry.

sensitive_parameters

when specified and permitted An object that contains arbitrary key/value pairs and describes the sensitive connection options for the entry.

Request and response examples

An empty request body.

```
{ }
```

A response for an empty request.

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["devices.arecool", "dont.youthink"],
      "type": "winrm",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    },
    {
      "connection_id": "4932bfe7-69c4-412f-b15c-ac0a7c2883f1",
      "certnames": ["managing.devices", "is.evencooler"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```


A certnames request body.

```
{
  "certnames": ["averygood.device"]
}
```

A response for a certnames request.

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```

A request to list a specific certname and its sensitive parameters: `/query/connections?sensitive=true`

```
{
  "certnames": ["averygood.device"],
  "extract": ["certnames", "sensitive_parameters"]
}
```

A response for a request for a specific certname and its sensitive parameters.

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["averygood.device"],
      "sensitive_parameters": {
        "username": "johnjohnjimmyjohn",
        "password": "C7b0$ls8ltO"
      }
    }
  ]
}
```

Query endpoint error responses

General format of error responses.

Every error response from the inventory service is a JSON response. Each response is an object that contains the following keys:

kind

The kind of error encountered.

msg

The message associated with the error.

details

A hash with more information about the error.

For example, if the request is missing the required content type headers, the following error occurs:

```
{
  "kind" : "puppetlabs.inventory/not-acceptable",
  "msg" : "accept must include content-type json",
  "details" : ""
}
```

kind error responses

For this endpoint, the `kind` key of the error displays the conflict.

puppetlabs.inventory/unknown-error

If an unknown error occurs during the request, the server returns a 500 response.

puppetlabs.inventory/not-acceptable

If the content provided to the API contains an "accepts" header which does not allow for JSON, the server returns a 406 response.

puppetlabs.inventory/unsupported-type

If the content provided to the API contains a "content-type" header other than JSON, the server returns a 416 response.

puppetlabs.inventory/json-parse-error

If there is an error while processing the request body, the server returns a 400 response.

puppetlabs.inventory/schema-validation-error

If there is a violation of the required format for the request body, the server returns a 400 response.

Managing patches

Use Puppet Enterprise to configure patching node groups to meet your needs, view available operating system patches for your nodes in the console, and apply patches using the `pe_patch::patch_server` task.

- [Configuring patch management](#) on page 450

To enable patch management, create a node group for nodes you want to patch and add the node group to the **PE Patch Management** parent node group.

- [Patching nodes](#) on page 456

After configuring patch management, you can start applying patches to nodes. The `patch_server` task enables simply applying patches, while the `group_patching` plan performs health checks before and after patches are applied.

Configuring patch management

To enable patch management, create a node group for nodes you want to patch and add the node group to the **PE Patch Management** parent node group.

Patch management OS compatibility

Patch management is compatible with select agent operating systems.

Note: If your operating system does not support TLSv1.2, see [Enable TLSv1](#) on page 686 to enable older protocols.

Operating system	Versions
CentOS	6, 7, 8
Oracle Linux	7, 8
Red Hat Enterprise Linux	7, 8
Scientific Linux	7
SUSE Linux Enterprise Server	12, 15
Ubuntu	16.04, 18.04, 20.04
Debian	9, 10
Fedora Note: You must install cron to run patch management on Fedora. To install cron, run <code>dnf install cronie</code>	30, 31
Microsoft Windows	10
Microsoft Windows Server Note: You must use Powershell 3.0 or higher to patch Windows nodes.	2012, 2012 R2, 2016, 2019

Where patch information comes from

Your package management software is responsible for ensuring PE can find the latest patch information available.

The `pe_patch` module uses OS level tools or APIs to find patches for nodes. You still have to manage the configuration of your package manager, like YUM, APT, Zypper, or Windows Update, so your nodes can search for updates. For example, if you need to go through a proxy and you use YUM, you must configure this on your own.

Note: To restrict which packages your OS finds and applies patches to, pin a package using `yum versionlock`, `apt-mark`, or `zypper addlock`, or with a package resource defined in the catalog for the node. The `pinned_packages` field in the `pe_patch` fact refers to versions locked using these methods; it does not refer to apt *pinned* packages, which prioritize packages, rather than locking them at a specific version.

Security updates

To find security updates, the `pe_patch` module uses security metadata when it is available. For example, Red Hat provides security metadata as additional metadata in YUM, Debian performs checks on the repo the updates are coming from, and Windows provides this information by default.

In the console, on the **Patches** page, security metadata feeds into the **Apply patches** table where you can filter for **Security updates only**.

Configure Windows Update

If you are using Windows Update, we recommend you use the [puppetlabs/wsus_client](#) module and configure these parameters in the `wsus_client` class.

- Set the `server_url` parameter to the URL of your WSUS server.
- Set the `auto_update_options` parameter to `AutoNotify` to automatically download updates and notify users.

Create a node group for nodes under patch management

Create a node group for nodes you want to patch in PE and add nodes to it. For example, create a node group for testing Windows and *nix patches prior to rolling out patches to other node groups. The **PE Patch Management** parent node group has the `pe_patch` class assigned to it and is in the console by default.

Note: Adding PE infrastructure nodes to patch management node groups can cause service interruptions when certain patches are applied.

1. In the console, click **Node groups**, and click **Add group**.
2. Specify options for the new node group, then click **Add**.
 - Parent name – Select **PE Patch Management**.
 - Group name – Enter a name that describes the role of the node group, for example, `patch test`.
 - Environment – Select **production**.
 - Environment group – Do not select this option.
3. Select the patching node group you created.
4. On the **Node group details** page, on the **Rules** tab, add nodes to the group by either pinning them individually or adding a rule to automatically add nodes that meet your specifications.



CAUTION: Do not include the same node in multiple node groups under patch management. This might cause classification conflicts.

5. Select **Run > Puppet** in the top right corner of the page.

PE can now manage patches for the nodes in your new node group. Repeat these steps to add any additional node groups you want under patch management.

Related information

[Add nodes to a node group](#) on page 319

There are two ways to add nodes to a node group.

Specify patching parameters

Set parameters for node groups under patch management by first applying the `pe_patch` class to them, then specifying your desired parameters.

Before you begin

Create at least one node group under patch management.

1. On the **Node groups** page, select the patching node group you want to add parameters to.
2. If it doesn't already exist, add the `pe_patch` class to the node group.
 - a) On the **Classes** tab, enter `pe_patch` and select **Add class**.
 - b) Commit changes.
3. In the `pe_patch` class, add the `patch_group` parameter and specify a value that describes the nodes in this node group.

Tip: The `patch_group` parameter is used to identify which nodes to run patching plans against. You might specify `patch_group` names that match your node groups, or apply the same `patch_group` parameter across several patching node groups that have similar characteristics.
4. Specify any additional patching parameters in the `pe_patch` class.
5. Commit changes.

Run Puppet on nodes in the node group before running patching tasks or plans.

Assign a patch management blackout window

Apply a blackout window to prevent PE from applying patches to nodes for a specified duration of time. For example, limit applying patches during an end-of-year change freeze.

Before you begin

Assign the `pe_patch` class to the applicable node group. See [Specify patching parameters](#) on page 452 for more information.

1. On the **Node groups** page, select the patching node group you want to assign a blackout window to.
2. On the **Classes** tab, under **Parameter**, add the `blackout_windows` parameter to the `pe_patch` class.
3. In the **Value** field, enter your blackout window as a JSON hash of keys and an ISO compliant timestamp.

For example, an end of year blackout window from the beginning of the day on 15 December 2020 to the end of the day on 15 January 2021 looks like this:

```
{
  "End of year change freeze": {
    "start": "2020-12-15T00:00:00+10:00",
    "end": "2021-01-15T23:59:59+10:00"
  }
}
```

4. Commit changes.

When a user tries to patch nodes during the blackout window, the **Patch blocked** field on the **Apply patches** table changes from **No** to **Yes** for affected patches. If the user proceeds with patching, the patching task fails.

Patch management parameters

Configure and tune patch management by adjusting parameters in the `pe_patch` class.

`patch_data_owner`

User name for the owner of the patch data. String.

Default: `root`

`patch_data_group`

Group name for the owner of the patch data. String.

Default: `root`

`patch_cron_user`

User who runs the cron job. String.

Default: `$patch_data_owner`

`manage_yum_utils`

Determines if the `yum_utils` package should be managed by this module on RedHat family nodes. If `true`, use the `yum_utils` parameter to determine how it should be managed. Boolean.

Default: `false`

`yum_utils`

If managed, determines what the package is set to. Enum[`installed`, `absent`, `purged`, `held`, `latest`]

Default: `installed`

`block_patching_on_warnings`

Determines if the patching task should run if there were warnings present on the `pe_patch` fact. If `true`, the run will abort and take no action. If `false`, the run will continue and attempt to patch. Boolean.

Default: `false`

fact_upload

Determines if puppet fact upload runs after any changes are made to the fact cache files. Boolean.

Default: true

apt_autoremove

Determines if apt-get autoremove runs during reboot. Boolean.

Default: false

manage_delta_rpm

Determines if the delta_rpm package should be managed by this module on RedHat family nodes. If true, use the delta_rpm parameter to determine how it should be managed. Boolean.

Default: false

delta_rpm

If managed, determines what the delta_rpm package is set to. Enum[installed, absent, purged, held, latest]

Default: installed

manage_yum_plugin_security

Determines if the yum_plugin_security package should be managed by this module on RedHat family nodes. If true, use the yum_plugin_security parameter to determine how it should be managed. Boolean.

Default: false

yum_plugin_security

If managed, determines what the yum_plugin_security package is set to. Enum[installed, absent, purged, held, latest]

Default: installed

reboot_override

Determines if a node reboots after patching. This overrides the setting in the task. Variant, Boolean, Enum[always, never, patched, smart, default]

- always - The node always reboots during the task run, even if no patches are required.
- never (or false) - The node never reboots during the task run, even if patches are applied.
- patched (or true) - The node reboots if patches are applied.
- smart - Use the OS supplied tools, like needs_restarting on RHEL or a pending reboot check on Windows, to determine if a reboot is required, if it is reboots, or if it does not reboot.
- default - Uses whatever option is set in the reboot parameter for the pe_patch::patch_server task.

Default: default

patch_group

Identifies nodes in or across patching node groups to run patching plans against.

Default: undef

pre_patching_scriptpath

The full path to an executable script or binary on the target node to be run before patching.

Default: undef

post_patching_scriptpath

The full path to an executable script or binary on the target node to be run after patching.

Default: undef

patch_cron_hour

The hour or hours for the cron job to run.

Default: absent, or *

patch_cron_month

The month or months for the cron job to run.

Default: absent, or *

patch_cron_monthday

The monthday or monthdays for the cron job to run.

Default: absent, or *

patch_cron_weekday

The weekday or weekdays for the cron job to run.

Default: absent, or *

patch_cron_min

The min or mins for the cron job to run.

Default: `fqdn_rand(59)` - a random number between 0 and 59.

ensure

Use `present` to install scripts, cronjobs, files, etc. Use `absent` to clean up system that previously hosted.

Default: `present`

blackout_windows

Determines a window of time when nodes cannot be patched. Hash.

`:title` - Name of the blackout window. String.

`:start` - Start of the blackout window (ISO8601 format). String.

`:end` - End of the blackout window (ISO8601 format). String.

Default: `undef`

windows_update_criteria

Determines which types of updates Windows Update searches for. To search both software and driver updates, remove the `Type` argument. String.

Default: `IsInstalled=0 and IsHidden=0 and Type='Software'`

Note: See the [Microsoft documentation](#) for more information about formatting strings for Windows Update.

Disable patch management

Use the console to disable patch management by editing the `ensure` parameter in the **PE Patch Management** node group. You can also remove patch management by deleting patching node groups.

1. In the console, click **Node groups** and select the **PE Patch Management** node group.
2. On the **Classes** tab, under the `pe_patch` class, select the `ensure` parameter, and change the value to `absent`.
3. Click **Add to node group** and commit the change.
4. Run Puppet.
The client components of the `pe_patch` class, like `cron` and `scripts`, are removed from PE.
5. Optional: To remove patch management from your infrastructure, click **Remove node group** on the **Node details** page for the **PE Patch Management** node group.

Note: If you have any child node groups under patch management, you must remove those node groups prior to removing the **PE Patch Management** parent node group.

The **Patch Management** section in the console sidebar remains active after disabling patch management, but the **Patches** page no longer reports patch information.

Patching nodes

After configuring patch management, you can start applying patches to nodes. The `patch_server` task enables simply applying patches, while the `group_patching` plan performs health checks before and after patches are applied.

Patch nodes

Use the `patch_server` task to apply patches to nodes. You can limit patches to security or non-security updates, Windows or *nix nodes, or a specific patch group.

Before you begin

Ensure you have permission to run the `pe_patch::patch_server` task.

1. On the **Patches** page, in the **Apply patches** section, use the filters to specify which patches to apply to which nodes.

Note: Filters use **and** logic. This means that if you select **Security updates** and **Windows**, the results include security patches for Windows nodes, not all security patches and all Windows patches.

2. Select **Run > Task**.

The **Run a task** page appears with patching information pre-filled for the `pe_patch::patch_server` task.

3. Optional: In the **Job details** field, provide a description of the task run. This text appears on the **Tasks** page.
4. Optional: Under **Task parameters**, add optional parameters to the task. See [Patching task parameters](#) on page 456 for a full list of available parameters.

Note: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

5. Optional: If you want to schedule the task to run later, under **Schedule**, select **Later** and choose a time.
6. Select **Run task** to apply patches.

To check the status of the task, look for it on the **Tasks** page. You can filter the results to view only `pe_patch` tasks.

Note: When using patch management to update core packages that affect the networking stack, the task run might look like it failed due to the PXP agent on the node losing connection with the primary server. However, the task still completes successfully. You can confirm by checking the `pe_patch` fact to verify the relevant packages were updated.

Patching task parameters

The `pe_patch::patch_server` task applies patches to nodes. When you patch nodes in the console, most of the information for the `patch_server` task is prefilled on the **Run a task** page, but you can add additional parameters to the task before you run it.

timeout

Indicates how much time elapses before the task run times out.

Accepted values: Any positive integer, in seconds.

Default: 3600

security_only

Indicates whether to apply only security patches.

Accepted values: `true`, `false`

Default: `false`

yum_params

Indicates additional parameters to include in yum commands, such as including or excluding repositories.

Accepted values: String

Default: undef

dpkg_params

Indicates additional parameters to include in apt-get commands.

Accepted values: String

Default: undef

zypper_params

Indicates additional parameters to include in zypper commands.

Accepted values: String

Default: undef

clean_cache

Indicates if yum or dpkg caches are cleaned at the start of the task.

Accepted values: true, false

Default: false

reboot

Indicates if and when nodes reboot during the task run.

Note: If the node group you're patching has a `reboot_override` value specified, that value overrides any `reboot` parameter you specify in task runs.

Accepted values:

- `always` — The node always reboots during the task run, even if no patches are required.
- `never` (or `false`) — The node never reboots during the task run, even if patches are applied.
- `patched` (or `true`) — The node reboots if patches are applied.
- `smart` — Use the OS supplied tools, like `needs_restarting` on RHEL or a pending reboot check on Windows, to determine if a reboot is required.

Default: never

Patch nodes with built-in health checks

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

Before you begin

Ensure you have permission to run the `pe_patch::group_patching` plan.

1. In the console, in the **Orchestration** section, select **Plans** and then click **Run a plan**.
2. Specify plan details:
 - **Code environment** — Select the environment where you installed the module containing the plan you want to run. For example, **production**.
 - **Job description** — Provide an optional description of the plan run.
 - **Plan** — Select `pe_patch::group_patching`.

3. In the **Plan parameters** section, specify the **patch_group** that you want to base running the plan on, and optionally add other patching plan parameters.

Note: The **patch_group** parameter is defined in the **pe_patch** class for node groups under patch management. For details, see [Specify patching parameters](#) on page 452.

4. Optional: In the **Schedule** section, specify if you want the plan to run at a later date and time.
5. Click **Run job**.

To check the status of the plan, look for it on the **Plans** page.

Note: When using patch management to update core packages that affect the networking stack, the task run might look like it failed due to the PXP agent on the node losing connection with the primary server. However, the task still completes successfully. You can confirm by checking the `pe_patch` fact to verify the relevant packages were updated.

Patching plan parameters

The `pe_patch::group_patching` plan verifies that Puppet is configured and running correctly on target nodes, patches the nodes, waits for any reboots, and then runs Puppet on the nodes to verify that they're still operational.

By default, the plan includes a health check which considers "healthy" any nodes on which:

- The Puppet service is enabled and running
- Noop mode and cached catalogs are not enabled
- The run interval is 30 minutes

You can modify plan behavior with several types of optional parameters:

- Patching options let you control how patching itself is applied, including adding an optional string to arguments passed to your package provider.
- Health check options control when a pre-patching health check and a post-patching Puppet run occurs.

Tip: The `health_check_*` parameters apply patches only to nodes that match values you specify. For example, if you change **health_check_service_running** to **false**, the pre-patching health check marks nodes on which the Puppet service *is* running as "unhealthy" and skips patching them.

- Reboot options control when a post-patching reboot occurs, and let you specify a script to execute after patching.

Patching options

patch_group

Specifies the **patch_group**, as defined in the **pe_patch** class parameter, that you want to base running the plan on.

Accepted values: String

patch_task_timeout

Indicates how much time elapses before the task run times out.

Accepted values: Any positive integer, in seconds.

Default: 3600

security_only

Indicates whether to apply only security patches.

Accepted values: `true`, `false`

Default: `false`

yum_params

Indicates additional parameters to include in yum commands, such as including or excluding repositories.

Accepted values: String

Default: `undef`

dpkg_params

Indicates additional parameters to include in apt-get commands.

Accepted values: String

Default: undef

zypper_params

Indicates additional parameters to include in zypper commands.

Accepted values: String

Default: undef

clean_cache

Indicates if yum or dpkg caches are cleaned at the start of the task.

Accepted values: true, false

Default: false

Health check options**run_health_check**

Indicates whether to do a pre-patching health check and a post-patching Puppet run.

Accepted values: true, false

Default: true

health_check_noop

Verifies the noop setting during pre-patching health checks.

Accepted values: true, false

Default: false

health_check_runinterval

Verifies the runinterval setting during pre-patching health checks.

Accepted values: Any positive integer, in seconds.

Default: 1800 (equivalent to the default Puppet run interval of 30 minutes)

health_check_service_running

Verifies whether the Puppet service is running during pre-patching health checks.

Accepted values: true, false

Default: true

health_check_service_enabled

Verifies whether the Puppet service is enabled during pre-patching health checks.

Accepted values: true, false

Default: true

health_check_use_cached_catalog

Verifies the use_cached_catalog setting during pre-patching health checks.

Accepted values: true, false

Default: false

Reboot options**reboot**

Indicates if and when nodes reboot during the plan run.

Note: If the node group you're patching has a `reboot_override` value specified, that value overrides any `reboot` parameter you specify in plan runs.

Accepted values:

- `always` — The node always reboots during the plan run, even if no patches are required.
- `never` — The node never reboots during the plan run, even if patches are applied.
- `patched` — The node reboots if patches are applied.
- `smart` — Use the OS supplied tools, like `needs_restarting` on RHEL or a pending reboot check on Windows, to determine if a reboot is required.

Default: `patched`

`reboot_wait_time`

Indicates how long to wait for nodes to reboot before running a post-patching health check.

Accepted values: Any positive integer, in seconds.

Default: 600

`post_reboot_scriptpath`

The full path to an executable script or binary on the target node to be run after reboot and before the final Puppet run.

Accepted values: File path

Default: `undef`

Orchestrating Puppet runs, tasks, and plans

Puppet orchestrator is an effective tool for making on-demand changes to your infrastructure.

With orchestrator you can initiate Puppet, task, or plan runs whenever you need them, eliminating manual work across your infrastructure.

- [How Puppet orchestrator works](#) on page 461

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

- [Setting up the orchestrator workflow](#) on page 463

The orchestrator—used alongside other PE tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

- [Configuring Puppet orchestrator](#) on page 469

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

- [Running Puppet on demand](#) on page 476

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

- [Tasks in PE](#) on page 489

Tasks are ad-hoc actions you can execute on a target and run from the command line or the console.

- [Plans in PE](#) on page 517

Plans allow you to tie together tasks, scripts, commands, and other plans to create complex workflows with refined access control. You can install modules that contain plans or write your own, then run them from the console or the command line.

- [Puppet orchestrator API v1 endpoints](#) on page 548

Use this API to gather details about the orchestrator jobs you run.

- [Migrating Bolt tasks and plans to PE](#) on page 611

If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a PE environment. This lets you manage and run tasks and plans using PE and

the console. PE environments and Bolt projects use the same default structure, and both use a Puppetfile to install content.

How Puppet orchestrator works

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

When you run Puppet on-demand with the orchestrator, you control the rollout of configuration changes when and how you want them. You control when Puppet runs and where node catalogs are applied (from the environment level to an individual node). You no longer need to wait on arbitrary run times to update your nodes.

Puppet tasks allow you to execute actions on target machines. A "task" is a single action that you execute on the target via an executable file. For example, do you want to upgrade a package or restart a particular service? Set up a Puppet task run to enforce to make those changes at will.

Puppet plans are bundles of tasks that can be combined with other logic. They allow you to do complex operations, like run multiple tasks with one command or automatically run certain tasks based on the output of another task.

Tasks and plans are packaged and distributed as Puppet modules.

Puppet orchestrator technical overview

The orchestrator uses pe-orchestration-services, a JVM-based service in PE, to execute on-demand Puppet runs on agent nodes in your infrastructure. The orchestrator uses PXP agents to orchestrate changes across your infrastructure.

The orchestrator (as part of pe-orchestration-services) controls the functionality for the `puppet - job` and `puppet - app` commands, and also controls the functionality for jobs and single node runs in the PE console.

The orchestrator is comprised of several components, each with their own configuration and log locations.

Puppet orchestrator architecture

The functionality of the orchestrator is derived from the Puppet Execution Protocol (PXP) and the Puppet Communications Protocol (PCP).

- PXP: A message format used to request that a task be executed on a remote host and receive responses on the status of that task. This is used by the pe-orchestration services to run Puppet on agents.
- PXP agent: A system service in the agent package that runs PXP.
- PCP: The underlying communication protocol that describes how PXP messages get routed to an agent and back to the orchestrator.
- PCP broker: A JVM-based service that runs in pe-orchestration-services on the primary server and in the pe-puppetserver service on compilers. PCP brokers route PCP messages, which declare the content of the message via message type, and identify the sender and intended recipient. PCP brokers on compilers connect to the orchestrator, and the orchestrator uses the brokers to direct messages to PXP agents connected to the compilers. When using compilers, PXP agents running on PE components (the primary server, PuppetDB, and the PE console) connect directly to the orchestrator, but all other PXP agents connect to compilers via load balancers.

What happens during an on-demand run from the orchestrator ?

Several PE services interact when you run Puppet on demand from the orchestrator.

1. You use the `puppet - job` command to create a job in orchestrator.
2. The orchestrator validates your token with the PE RBAC service.
3. The orchestrator requests environment classification from the node classifier for the nodes targeted in the job, and it queries PuppetDB for the nodes.
4. The orchestrator requests the environment graph from Puppet Server.
5. The orchestrator creates the job ID and starts polling nodes in the job to check their statuses.
6. The orchestrator queries PuppetDB for the agent version on the nodes targeted in the job.

7. The orchestrator tells the PCP broker to start runs on the nodes targeted in the job, and Puppet runs start on those agents.
8. The agent sends its run results to the PCP broker.
9. The orchestrator receives run results, and requests the node run reports from PuppetDB.

What happens during a task run from the orchestrator?

Several services interact for a task run as well. Because tasks are Puppet code, they must be deployed into an environment on the primary server. Puppet Server then exposes the task metadata to the orchestrator. When a task is run, the orchestrator sends the PXP agent a URL of where to fetch the task from the primary server and the checksum of the task file. The PXP agent downloads the task file from the URL and caches it for future use. The file is validated against the checksum before every execution. The following are the steps in this process.

1. The PE client sends a task command.
2. The orchestrator checks if a user is authorized.
3. The orchestrator fetches the node target from PuppetDB if the target is a query, and returns the nodes.
4. The orchestrator requests task data from Puppet Server.
5. Puppet Server returns task metadata, file URIs, and file SHAs.
6. The orchestrator validates the task command and then sends the job ID back to the client.
7. The orchestrator sends task parameters and file information to the PXP agent.
8. The PXP agent sends a provisional response to the orchestrator, checks the SHA against the local cache, and requests the task file from Puppet Server.
9. Puppet Server returns the task file to the PXP agent.
10. The task runs.
11. The PXP agent sends the result to the orchestrator.
12. The client requests events from the orchestrator.
13. The orchestrator returns the result to the client.

Configuration notes for the orchestrator and related components

Configuration and tuning for the components in the orchestrator happens in various files.

- `pe-orchestration-services`: The underlying service for the orchestrator. The main configuration file is `/etc/puppetlabs/orchestration-services/conf.d`.

Additional configuration for large infrastructures can include tuning the `pe-orchestration-services` JVM heap size, increasing the limit on open file descriptors for `pe-orchestration-services`, and tuning ARP tables.

- `PCP broker`: Part of the `pe-puppetserver` service. The main configuration file is `/etc/puppetlabs/puppetserver/conf.d`.

The PCP broker requires JVM memory and file descriptors, and these resources scale linearly with the number of active connections. Specifically, the PCP broker requires:

- Approximately 40 KB of memory (when restricted with the `-Xmx` JVM option)`
- One file descriptor per connection
- An approximate baseline of 60 MB of memory and 200 file descriptors

For a deployment of 100 agents, expect to configure the JVM with at least `-Xmx64m` and 300 file descriptors. Message handling requires minimal additional memory.

- `PXP agent`: Configuration is managed by the agent profile (`puppet_enterprise::profile::agent`).

The PXP agent is configured to use Puppet's SSL certificates and point to one PCP broker endpoint. If disaster recovery is configured, the agent points to additional PCP broker endpoints in the case of failover.

Note: If you reuse an existing agent with a new orchestrator instance, you must delete the `pxp-agent` spool directory, located at `/opt/puppetlabs/pxp-agent/spool` (*nix) or `C:\ProgramData\PuppetLabs\pxp-agent\var\spool` (Windows)

Debugging the orchestrator and related components

If you need to debug the orchestrator or any of its related components, the following log locations might be helpful.

- **pe-orchestration-services:** The main log file is `/var/log/puppetlabs/orchestration-services/orchestration-services.log`.
- **PCP:** The main log file for PCP brokers on compilers is `/var/log/puppetlabs/puppetserver/pcp-broker.log`. You can configure logback through the Puppet server configuration.

The main log file for PCP brokers on the primary server is `/var/log/puppetlabs/orchestration-services/pcp-broker.log`.

You can also enable an access log for messages.

- **PXP agent:** The main log file is `/var/log/puppetlabs/pxp-agent/pxp-agent.log` (on *nix) or `C:\ProgramData\PuppetLabs\pxp-agent\var\log\pxp-agent.log` (on Windows). You can configure this location as necessary.

Additionally, metadata about Puppet runs triggered via the PXP agent are kept in the spool-dir, which defaults to `/opt/puppetlabs/pxp-agent/spool` (on *nix) and `C:\ProgramData\PuppetLabs\pxp-agent\var\spool` (on Windows). Results are kept for 14 days.

Setting up the orchestrator workflow

The orchestrator—used alongside other PE tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

This recommended workflow gives you precise control over rolling out changes, from updating data and classifying nodes, to deploying new Puppet code. You configure your agents to use cached catalogs during scheduled runs, and send new catalogs only when you're ready via orchestrator jobs. Scheduled runs continue to enforce the desired state of the last orchestration job until you send another new catalog.

- To use this workflow, you must enable cached catalogs for use with the orchestrator so that they enforce cached catalogs by default and compile new catalogs only when instructed to by orchestrator jobs.
- This workflow also assumes you're familiar with Code Manager. It involves making changes to your control repo—adding or updating modules, editing manifests, or changing your Hiera data. You'll also run deploy actions from the Code Manager command line tool and the orchestrator, so ensure you have access to a host with PE client tools installed.

Related information

[Enable cached catalogs for use with the orchestrator \(optional\)](#)

Set up node groups for testing new features

The first step in the workflow is to set up node groups for testing your new feature or code.

1. If they don't already exist, create environment node groups for branch testing, for example, you might create `Development environment` and `Test environment` node groups.
2. Within each of these environment node groups, create a child node group to enable on-demand testing of changes deployed in Git feature branch Puppet environments.

You now have at three levels of environment node groups: 1) the top-level parent environment node group, 2) node groups that represent your actual environments, and 3) node groups specific to feature testing.

3. In the **Rules** tab of the child node groups you created in the previous step, add this rule:

Option	Value
Fact	agent_specified_environment
Operator	~
Value	^.+

This rule matches any nodes from the parent group that have the **agent_specified_environment** fact set. By matching nodes to this group, you give the nodes permission to override the server-specified environment and use their agent-specified environment instead.

Related information

[Create environment node groups](#) on page 318

Create custom environment node groups so that you can target deployment of Puppet code.

Create a feature branch

After you've set up a node group, create a new branch of your control repository on which you can make changes to your feature code.

1. Branch your control repository, and name the new branch, for example, `my_feature_branch`.

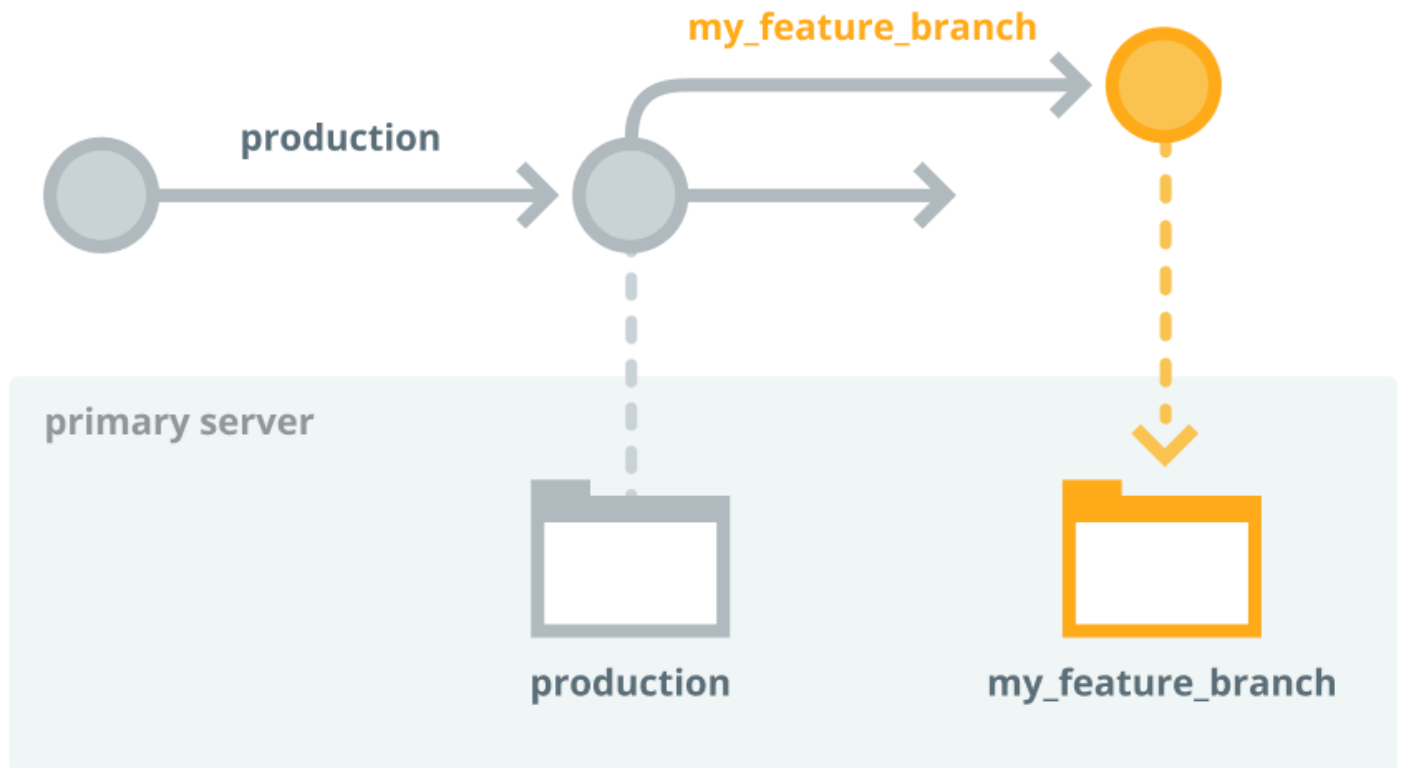


2. Make changes to the code on your feature branch. Commit and push the changes to the `my_feature_branch`.

Deploy code to the primary server and test it

Now that you've made some changes to the code on your feature branch, you're ready to use Code Manager to push those to the primary server.

1. To deploy the feature branch to the primary server, run the following Code Manager command: `puppet code deploy --wait my_feature_branch`

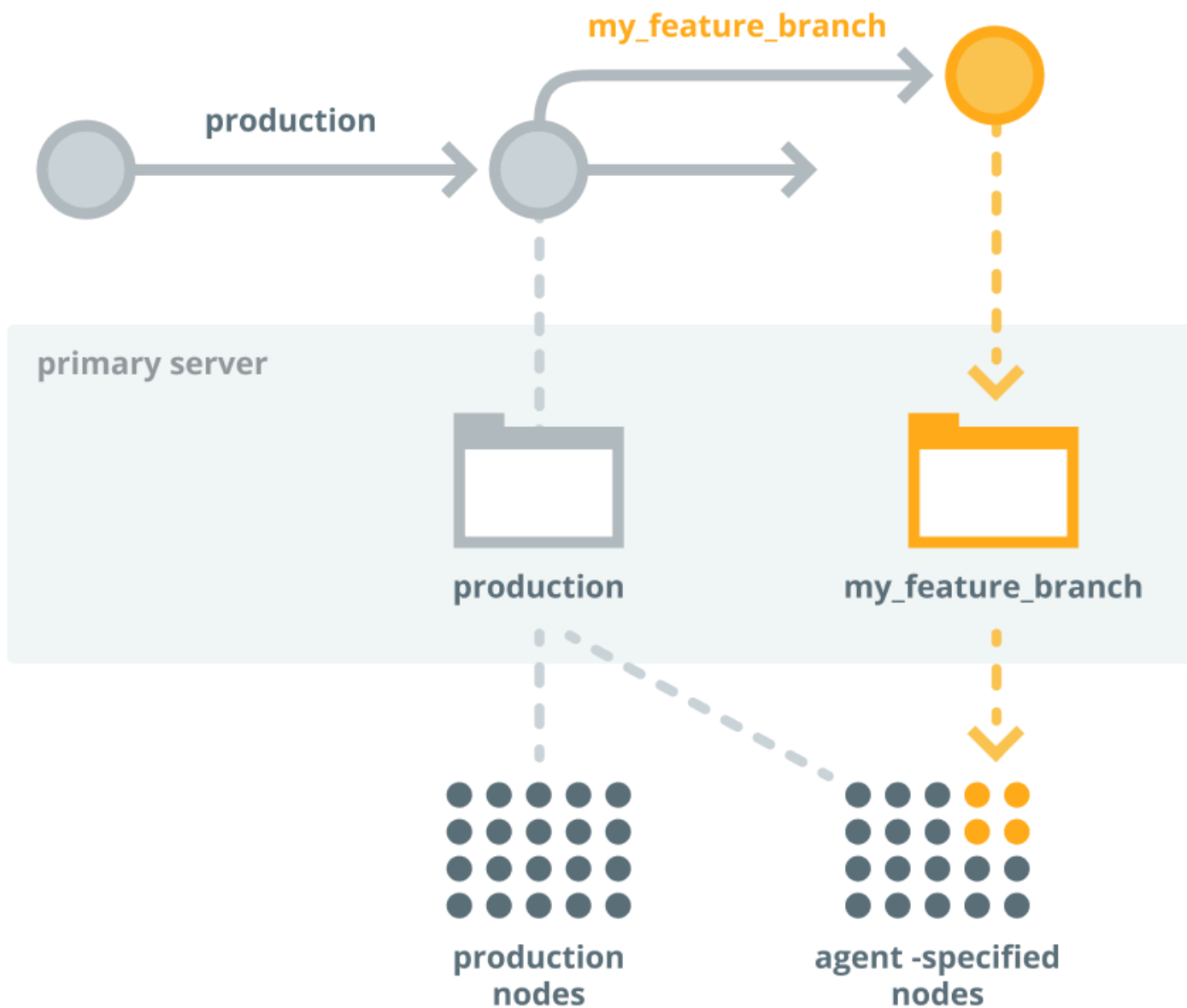


Note: After this code deployment, there is a short delay while Puppet Server loads the new code.

- To test your changes, run Puppet on a few agent-specified development nodes in the **my_feature_branch** environment, run the following orchestrator command:

```
puppet job run --nodes my-dev-node1,my-dev-node2 --environment my_feature_branch
```

Tip: You can also use the console to create a job targeted at a list of nodes in the **my_feature_branch** environment.



- Validate your testing changes. Open the links in the orchestrator command output, or use the Job ID linked on the Job list page, to review the node run reports in the console. Ensure the changes have the effect you intend.

Related information

[Run Puppet on a node list](#) on page 476

Create a node list target for a job when you need to run Puppet on a specific set of nodes that isn't easily defined by a PQL query.

Merge and promote your code

If everything works as expected on the development nodes, and you're ready to promote your changes into production.

1. Merge `my_feature_branch` into the `production` branch in your control repo.



2. To deploy your updated `production` branch to the primary server, run the following Code Manager command:
`puppet code deploy --wait production`

Preview the job

Before running Puppet across the `production` environment, preview the job with the `puppet job plan` command.

To ensure the job captures all the nodes in the `production` environment, as well as the agent-specified development nodes that just ran with the `my_feature_branch` environment, use the following query as the job target:

```
puppet job plan --query 'inventory {environment in ["production",
"my_feature_branch"]}'
```

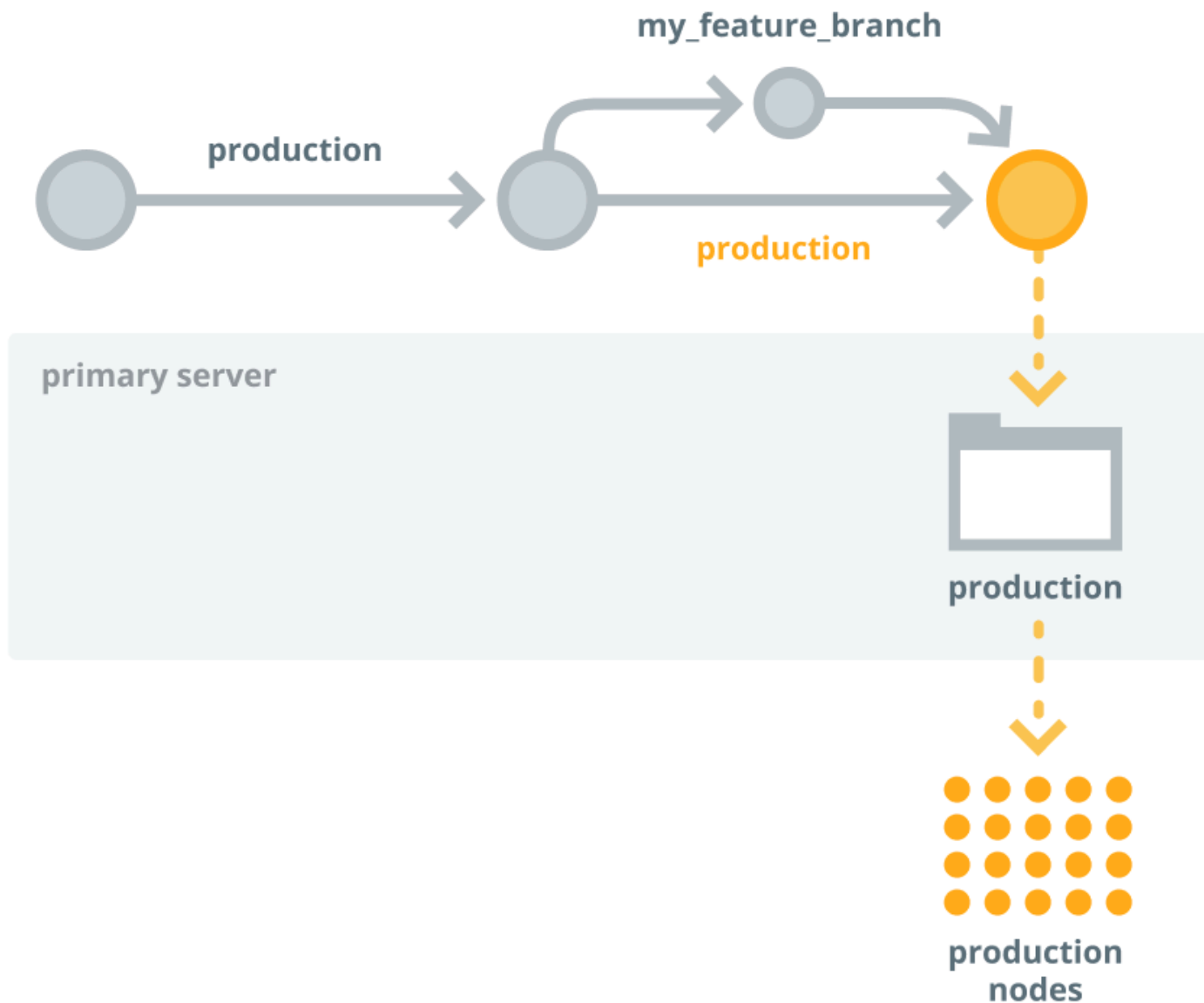
Run the job on the production environment

If you're satisfied with the changes in the preview, you're ready to enforce changes to the `production` environment.

Run the orchestrator job.

```
puppet job run --query 'inventory {environment in ["production",
"my_feature_branch"]}'
```

Tip: You can also use the console to create a job targeted at this PQL query.



Related information

[Run Puppet on a PQL query](#) on page 477

For some jobs, you might want to target nodes that meet specific conditions. For such jobs, create a PQL query.

Validate your production changes

Finally, you're ready to validate your production changes.

Check the node run reports in the console to confirm that the changes were applied as intended. If so, you're done!

Repeat this process as you develop and promote your code.

Configuring Puppet orchestrator

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

- Set PE RBAC permissions and token authentication for Puppet orchestrator
- Enable cached catalogs for use with the orchestrator (optional)
- Review the orchestrator configuration files and adjust them as needed

All of these instructions assume that PE client tools are installed.

Related information

[Installing PE client tools](#) on page 138

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

Orchestration services settings

global.conf: Global logging and SSL settings

`/etc/puppetlabs/orchestration-services/conf.d/global.conf` contains settings shared across the Puppet Enterprise (PE) orchestration services.

The file `global.certs` typically requires no changes and contains the following settings:

Setting	Definition	Default
<code>ssl-cert</code>	Certificate file path for the orchestrator host.	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.cert.pem</code>
<code>ssl-key</code>	Private key path for the orchestrator host.	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.private_key.pem</code>
<code>ssl-ca-cert</code>	CA file path	<code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code>

The file `global.logging-config` is a path to `logback.xml` file that configures logging for most of the orchestration services. See <http://logback.qos.ch/manual/configuration.html> for documentation on the structure of the `logback.xml` file. It configures the log location, rotation, and formatting for the following:

- `orchestration-services` (appender section F1)
- `orchestration-services status` (STATUS)
- `pcp-broker` (PCP)
- `pcp-broker access` (PCP_ACCESS)
- `aggregate-node-count` (AGG_NODE_COUNT)

bootstrap.cfg: Allow list of trapperkeeper services to start

`/etc/puppetlabs/orchestration-services/bootstrap.cfg` is the list of trapperkeeper services from the orchestrator and pcp-broker projects that are loaded when the `pe-orchestration-services` system service starts.

- To disable a service in this list, remove it or comment it with a `#` character and restart `pe-orchestration-services`

- To enable an NREPL service for debugging, add `puppetlabs.trapperkeeper.services.nrepl.nrepl-service/nrepl-service` to this list and restart `pe-orchestration-services`.

webserver.conf and web-routes.conf: The pcp-broker and orchestrator HTTP services

`/etc/puppetlabs/orchestration-services/conf.d/webserver.conf` describes how and where to the run `pcp-broker` and `orchestrator` web services, which accept HTTP API requests from the rest of the PE installation and from external nodes and users.

The file `webserver.orchestrator` configures the `orchestrator` web service. Defaults are as follows:

Setting	Definition	Default
<code>access-log-config</code>	A logback XML file configuring logging for orchestrator access messages.	<code>/etc/puppetlabs/orchestration-services/request-logging.xml</code>
<code>client-auth</code>	Determines the mode that the server uses to validate the client's certificate for incoming SSL connections.	want or need
<code>default-server</code>	Allows multi-server configurations to run operations without specifying a server-id. Without a server-id, operations will run on the selected default. Optional.	true
<code>ssl-ca-cert</code>	Sets the path to the CA certificate PEM file used for client authentication.	<code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code>
<code>ssl-cert</code>	Sets the path to the server certificate PEM file used by the web service for HTTPS.	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.cert.pem</code>
<code>ssl-crl-path</code>	Describes a path to a Certificate Revocation List file. Optional.	<code>/etc/puppetlabs/puppet/ssl/crl.pem</code>
<code>ssl-host</code>	Sets the host name to listen on for encrypted HTTPS traffic.	0.0.0.0
<code>ssl-key</code>	Sets the path to the private key PEM file that corresponds with the <code>ssl-cert</code>	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.private_key.pem</code>
<code>ssl-port</code>	Sets the port to use for encrypted HTTPS traffic.	8143

The file `webserver.pcp-broker` configures the `pcp-broker` web service. Defaults are as follows:

Setting	Definition	Default
<code>client-auth</code>	Determines the mode that the server uses to validate the client's certificate for incoming SSL connections.	want or need
<code>ssl-ca-cert</code>	Sets the path to the CA certificate PEM file used for client authentication.	<code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code>

Setting	Definition	Default
<code>ssl-cert</code>	Sets the path to the server certificate PEM file used by the web service for HTTPS.	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.cert.pem</code>
<code>ssl-crl-path</code>	Describes a path to a Certificate Revocation List file. Optional.	<code>/etc/puppetlabs/puppet/ssl/crl.pem</code>
<code>ssl-host</code>	Sets the host name to listen on for encrypted HTTPS traffic.	<code>0.0.0.0</code> .
<code>ssl-key</code>	Sets the path to the private key PEM file that corresponds with the <code>ssl-cert</code> .	<code>/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.private_key.pem</code>
<code>ssl-port</code>	Sets the port to use for encrypted HTTPS traffic.	<code>8142</code>

`/etc/puppetlabs/orchestration-services/conf.d/web-routes.conf` describes how to route HTTP requests made to the API web servers, designating routes for interactions with other services. These should not be modified. See the configuration options at the [trapperkeeper-webserver-jetty project's docs](#)

analytics.conf: Analytics trapperkeeper service configuration

`/etc/puppetlabs/orchestration-services/conf.d/analytics.conf` contains the internal setting for the [analytics](#) trapperkeeper service.

Setting	Definition	Default
<code>analytics.url</code>	Specifies the API root.	<code><puppetserver-host-url>:8140/analytics/v1</code>

auth.conf: Authorization trapperkeeper service configuration

`/etc/puppetlabs/orchestration-services/conf.d/auth.conf` contains internal settings for the authorization trapperkeeper service. See configuration options in the [trapperkeeper-authorization project's docs](#).

metrics.conf: JXM metrics trapperkeeper service configuration

`/etc/puppetlabs/orchestration-services/conf.d/metrics.conf` contains internal settings for the JXM metrics service built into orchestration-services. See the service configuration options in the [trapperkeeper-metrics project's docs](#).

orchestrator.conf: Orchestrator trapperkeeper service configuration

`/etc/puppetlabs/orchestration-services/conf.d/orchestrator.conf` contains internal settings for the orchestrator project's trapperkeeper service.

pcp-broker.conf: PCP broker trapperkeeper service configuration

`/etc/puppetlabs/orchestration-services/conf.d/pcp-broker.conf` contains internal settings for the pcp-broker project's trapperkeeper service. See the service configuration options in the [pcp-broker project's docs](#).

Orchestrator configuration files

The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate work station, there are two types of configuration files: a global configuration file and a user-specified configuration file.

Orchestrator global configuration file

If you're running the orchestrator from a PE-managed machine, on either the primary server or an agent node, PE manages the global configuration file.

This file is installed on both managed and non-managed workstations at:

- ***nix systems** --- `/etc/puppetlabs/client-tools/orchestrator.conf`
- **Windows** --- `C:/ProgramData/PuppetLabs/client-tools/orchestrator.conf`

The class that manages the global configuration file is `puppet_enterprise::profile::controller`. The following parameters and values are available for this class:

Parameter	Value
<code>manage_orchestrator</code>	true or false (default is true)
<code>orchestrator_url</code>	url and port (default is primary server url and port 8143)

The only value PE sets in the global configuration file is the `orchestrator_url` (which sets the orchestrator's `service-url` in `/etc/puppetlabs/client-tools/orchestrator.conf`).

Important: If you're using a managed workstation, do not edit or change the global configuration file. If you're using an unmanaged workstation, you can edit this file as needed.

Orchestrator user-specified configuration file

You can manually create a user-specified configuration file and populate it with orchestrator configuration file settings. PE does not manage this file.

This file needs to be located at `~/.puppetlabs/client-tools/orchestrator.conf` for both *nix and Windows.

If present, the user specified configuration always takes precedence over the global configuration file. For example, if both files have contradictory settings for the **environment**, the user specified settings prevail.

Orchestrator configuration file settings

The orchestrator configuration file is formatted in JSON. For example:

```
{
  "options" : {
    "service-url": "https://<PRIMARY SERVER HOSTNAME>:8143",
    "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
    "token-file": "~/.puppetlabs/token",
    "color": true
  }
}
```

The orchestrator configuration files (the user-specified or global files) can take the following settings:

Setting	Definition
service-url	The URL that points to the primary server and the port used to communicate with the orchestration service. (You can set this with the <code>orchestrator_url</code> parameter in the <code>puppet_enterprise::profile::controller</code> class.) Default value: <code>https://<PRIMARY_SERVER_HOSTNAME>:8143</code>
environment	The environment used when you issue commands with Puppet orchestrator.
cacert	The path for the Puppet Enterprise CA cert. <ul style="list-style-type: none"> *nix: <code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code> Windows: <code>C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem</code>
token-file	The location for the authentication token. Default value: <code>~/.puppetlabs/token</code>
color	Determines whether the orchestrator output uses color. Set to <code>true</code> or <code>false</code> .
noop	Determines whether the orchestrator runs the Puppet agent in no-op mode. Set to <code>true</code> or <code>false</code> .

Setting PE RBAC permissions and token authentication for orchestrator

Before you run any orchestrator jobs, you need to set the appropriate permissions in PE role-based access control (RBAC) and establish token-based authentication.

Most orchestrator users require the following permissions to run orchestrator jobs or tasks:

Type	Permission	Definition
Puppet agent	Run Puppet on agent nodes.	The ability to run Puppet on nodes using the console or orchestrator. Instance must always be <code>"*"</code> .
Job orchestrator	Start, stop and view jobs	The ability to start and stop jobs and tasks, view jobs and job progress, and view an inventory of nodes that are connected to the PCP broker.
Tasks	Run tasks	The ability to run specific tasks on all nodes, a selected node group, or nodes that match a PQL query.
Nodes	View node data from PuppetDB.	The ability to view node data imported from PuppetDB. Object must always be <code>"*"</code> .

Note: If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

Assign task permissions to a user role

1. In the console, on the **Access control** page, click the **User roles** tab.
2. From the list of user roles, click the one you want to have task permissions.
3. On the **Permissions** tab, in the **Type** box, select **Tasks**.
4. For **Permission**, select **Run tasks**, and then select a task from the **Object** list. For example, **factor_task**.
5. Click **Add permission**, and then commit the change.

Using token authentication

Before running an orchestrator job, you must generate an RBAC access token to authenticate to the orchestration service. If you attempt to run a job without a token, PE prompts you to supply credentials.

For information about generating a token with the CLI, see the documentation on token-based authentication.

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

[Create a new user role](#) on page 202

RBAC has four predefined roles: Administrators, Code Deployers, Operators, and Viewers. You can also define your own custom user roles.

[Assign permissions to a user role](#) on page 202

You can mix and match permissions to create custom user roles that provide users with precise levels of access to PE actions.

PE Bolt services configuration

The PE Bolt server is a Ruby service that enables SSH and WinRM tasks from the console. The server accepts requests from the orchestrator, calls out to Bolt to run the task, and returns the result.

PE Bolt server configuration

The PE Bolt server provides an API for running tasks over SSH and WinRM using Bolt, which is the technology underlying PE tasks. You do not need to have Bolt installed to configure the Bolt server or run tasks in PE. The API server for tasks is available as `pe-bolt-server`.

The server is a [Puma](#) application that runs as a standalone service.

The server is configured in `/etc/puppetlabs/bolt-server/conf.d/bolt-server.conf`, managed by the `puppet_enterprise::profile::bolt_server` class, which includes these parameters:

Setting	Type	Description	Default
<code>bolt_server_loglevel</code>	string	Bolt log level. Acceptable values are debug, info, notice, warn, and error.	notice
<code>concurrency</code>	integer	Maximum number of server threads.	100
<code>master_host</code>	string	URI of the primary server where Bolt can download tasks.	<code>\$puppet_enterprise::puppet_ma</code>
<code>master_port</code>	integer	Port the Bolt server can access the primary server on.	<code>\$puppet_enterprise::puppet_ma</code>

Setting	Type	Description	Default
ssl_cipher_suites	array[string]	TLS cipher suites in order of preference.	<code>\$puppet_enterprise::params::ssl_cipher_suites</code>
ssl_listen_port	integer	Port the Bolt server runs on.	62658 (<code>\$puppet_enterprise::bolt_server_port</code>)
allowlist	array[string]	List of hosts that can connect to pe-bolt-server.	[<code>\$certname</code>]

PE ACE services configuration

The PE Agentless Catalog Executor server is a Ruby service that enables you to execute Bolt Tasks and Puppet catalogs remotely.

PE ACE server configuration

The PE ACE server provides an API for running tasks and catalogs against remote targets.

The server is a [Puma](#) application that runs as a standalone service.

The server is configured in `/etc/puppetlabs/ace-server/conf.d/ace-server.conf`, managed by the `puppet_enterprise::profile::ace_server` class, which includes these parameters:

Setting	Type	Description	Default
service_loglevel	string	Bolt log level. Acceptable values are debug, info, notice, warn, and error.	notice
concurrency	integer	Maximum number of server threads.	<code>\$puppet_enterprise::ace_server_threads</code>
master_host	string	URI that ACE can access the primary server on.	<code>pe_repo::compile_master_pool_host</code> default: <code>\$puppet_enterprise::puppet_master_host</code>
master_port	integer	Port that ACE can access the primary server on.	<code>\$puppet_enterprise::puppet_master_port</code>
hostcrl	string	The host CRL path	<code>\$puppet_enterprise::params::hostcrl</code>
ssl_cipher_suites	array[string]	TLS cipher suites in order of preference.	<code>\$puppet_enterprise::params::ssl_cipher_suites</code>
ssl_listen_port	integer	Port that ACE runs on.	44633 (<code>\$puppet_enterprise::ace_server_port</code>)
allowlist	array[string]	List of hosts that can connect to pe-ace-server.	[<code>\$certname</code>]

Disabling application management or orchestration services

Both application management and orchestration services are on by default in PE. If you need to disable these services, refer to [Disabling application management](#) and [Disabling orchestration services](#).

Running Puppet on demand

The orchestrator gives you the ability to set up jobs in the console or on the command line to trigger on-demand Puppet runs.

- [Running Puppet on demand from the console](#) on page 476

When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

- [Running Puppet on demand from the CLI](#) on page 482

Use the **puppet job run** command to enforce change on your agent nodes with on-demand Puppet runs.

- [Running Puppet on demand with the API](#) on page 486

Run the orchestrator across all nodes in an environment.

Running Puppet on demand from the console

When you set up a job to run Puppet from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs Puppet on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

There are two ways to specify the job target (the nodes you want to run jobs on):

- A static node list
- A Puppet Query Language (PQL) query
- A node group

You can't combine these methods, and if you switch from one to the other with the **Inventory** drop-down list, the target list clears and starts over. In other words, if you create a target list by using a node list, switching to a PQL query clears the node list, and vice versa. You can do a one-time conversion of a PQL query to a static node list if you want to add or remove nodes from the query results.

Before you start, be sure you have the correct permissions for running jobs. To run jobs on PQL queries, you need the "View node data from PuppetDB" permission.

Run Puppet on a node list

Create a node list target for a job when you need to run Puppet on a specific set of nodes that isn't easily defined by a PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run jobs and PQL queries.

Tip: You can add network devices to a node list when you have installed modules for device transports in your production environment. You can find such modules in [Puppet Forge](#).

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.

4. Select an environment:

- **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
- **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

Note: If your job target includes application instances, the selected environment also determines the dependency order of your node runs.

5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:

- **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
- **Debug:** Print all debugging messages.
- **Trace:** Print stack traces on some errors.
- **Eval-trace:** Display how long it took for each step to run.
- **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforces a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.

6. In the **Inventory** list, select **Node list**.

7. To create a node list, in the search field, start typing in the names of nodes to search for, and click **Search**.

Note: The search does not handle regular expressions.

8. Select the nodes you want to add to the job. You can select nodes from multiple searches to create the node list target.

9. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only on the nodes that failed during the initial run.

Run Puppet on a PQL query

For some jobs, you might want to target nodes that meet specific conditions. For such jobs, create a PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run jobs and PQL queries.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
4. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

Note: If your job target includes application instances, the selected environment also determines the dependency order of your node runs.

5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
 - **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
 - **Debug:** Print all debugging messages.
 - **Trace:** Print stack traces on some errors.
 - **Eval-trace:** Display how long it took for each step to run.
 - **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforces a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.
6. From the list of target types, select **PQL query**.
7. Specify a target by doing one of the following:
 - Enter a query that selects the target you want. See the [Puppet Query Language \(PQL\) reference](#) for more information.
 - Click **Common queries**. Select one of the queries and replace the defaults in the braces (`{ }`) with values that specify the target you want.

Note: These queries include `[certname]` as `[<projection>]` to restrict the output.

Target	PQL query
All nodes	<code>nodes[certname] { }</code>
Nodes with a specific resource (example: httpd)	<code>resources[certname] { type = "Service" and title = "httpd" }</code>
Nodes with a specific fact and value (example: OS name is CentOS)	<code>inventory[certname] { facts.os.name = "<OS>" }</code>
Nodes with a specific report status (example: last run failed)	<code>reports[certname] { latest_report_status = "failed" }</code>
Nodes with a specific class (example: Apache)	<code>resources[certname] { type = "Class" and title = "Apache" }</code>
Nodes assigned to a specific environment (example: production)	<code>nodes[certname] { catalog_environment = "production" }</code>
Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)	<code>resources[certname] { type = "Package" and title="openssl" and parameters.ensure = "1.0.1e-51.el7_2.7" }</code>
Nodes with a specific resource and operating system (example: httpd and CentOS)	<code>inventory[certname] { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }</code>

8. Click **Submit query** and click **Refresh** to update the node results.
9. If you change or edit the query after it runs, click **Submit query** again.
10. Optional: To convert the PQL query target results to a node list, for use as a node list target, click **Convert query to static node list**.

Note: If you select this option, the job target becomes a node list. You can add or remove nodes from the node list before running the job, but you cannot edit the query.

11. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only on the nodes that failed during the initial run.

Important: When you run this job, the PQL query runs again, and the job might run on a different set of nodes than what is currently displayed. If you want the job to run only on the list as displayed, convert the query to a static node list before you run the job.

Add custom PQL queries to the console

Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary sever, as root, copy the `custom_pql_queries.json.example` file and remove the `example` suffix.

```
cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

2. Edit the file contents to include your own PQL queries or remove any existing queries.
3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop down options when running jobs.

Run Puppet on a node group

Create a node target for a job when you need to run Puppet on a specific set of nodes in a pre-defined group.

Before you begin

Make sure you have access to the nodes you want to target.

Note: If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
4. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

Note: If your job target includes application instances, the selected environment also determines the dependency order of your node runs.

5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
 - **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
 - **Debug:** Print all debugging messages.
 - **Trace:** Print stack traces on some errors.
 - **Eval-trace:** Display how long it took for each step to run.
 - **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforces a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.
6. From the list of target types, select **Node group**.
7. In the **Choose a node group** box, type or select a node group, and click **Select**.

8. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun the it on all nodes or only on the nodes that failed during the initial run.

Run jobs throughout the console

You don't need to be in the Jobs section of the console to run a Puppet job on your nodes. You might encounter situations where you want to run jobs on lists of nodes derived from different pages in the console.

You can create jobs from the following pages:

Page	Description
Overview	This page shows a list of all your managed nodes, and gathers essential information about your infrastructure at a glance.
Events	Events let you view a summary of activity in your infrastructure, analyze the details of important changes, and investigate common causes behind related events. For instance, let's say you notice run failures because some nodes have out-of-date code. After you update the code, you can create a job target from the list of failed nodes to be sure you're directing the right fix to the right nodes. You can create new jobs from the Nodes with events category.
Classification node groups	Node groups are used to automate classification of nodes with similar functions in your infrastructure. If you make a classification change to a node group, you can quickly create a job to run Puppet on all the nodes in that group, pushing the change to all nodes at one time.

Make sure you have permissions to run jobs and PQL queries.

1. In the console, in the **Run** section, click **Puppet**.

At this point, the list of nodes is converted to a new Puppet run job list target.

2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

3. Select an environment:

- **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their puppet . conf file.
- **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

Note: If your job target includes application instances, the selected environment also determines the dependency order of your node runs.

4. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:

- **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
- **Debug:** Print all debugging messages.
- **Trace:** Print stack traces on some errors.
- **Eval-trace:** Display how long it took for each step to run.
- **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their puppet . conf files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.

5. **Important:** Do not change the Inventory from **Node list** to **PQL query**. This clears the node list target.

6. Click **Run job**.

View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun the it on all nodes or only on the nodes that failed during the initial run.

Schedule a Puppet run

Schedule a job to deploy configuration changes at a particular date and time.

Before you begin

Make sure you have access to the nodes you want to target.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Select an environment:

- **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
- **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified, if they're included in an application in that environment, or if they're classified in that environment by the node manager.

Note: If your job target includes application instances, the selected environment also determines the dependency order of your node runs.

4. Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
5. Optional: To repeat the job on a regular schedule, change the run frequency from **Once** to **Hourly**, **Daily**, or **Weekly**.

Note: If a recurring job runs longer than the selected frequency, the following job waits to start until the next available time interval.

6. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:

- **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
- **Debug:** Print all debugging messages.
- **Trace:** Print stack traces on some errors.
- **Eval-trace:** Display how long it took for each step to run.
- **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.

7. From the list of target types, select the category of nodes you want to target.

- **Node list** Add individual nodes by name.
- **PQL Query** Use the Puppet query language to retrieve a list of nodes.
- **Node group** Select an existing node group.

8. Click **Schedule job**.

Your job appears on the **Scheduled Puppet run** tab of the **Jobs** page.

Stop a job in progress

You can stop a job if, for example, you realize you need to reconfigure a class or push a configuration change that the job needs.

When you stop a Puppet job, jobs that are underway finish, and jobs that have not started are canceled.

To stop a job:

- In the console, go to the **Jobs** page and select the **Puppet run** tab. From the list of jobs, find the one you want and click **Stop**.
- On the command line, press **CTRL + C**.

Delete a scheduled job

Delete a job that is scheduled to run at a later time.

Tip: You must have the appropriate role-based permissions to delete another user's scheduled job.

1. In the console, open the **Jobs** page.
2. Click the **Scheduled Puppet Run** tab.
3. From the list of jobs, find the one you want to delete and click **Remove**.

Running Puppet on demand from the CLI

Use the **puppet job run** command to enforce change on your agent nodes with on-demand Puppet runs.

Use the `puppet job run` command to enforce change across nodes. For example, when you add a new class parameter to a set of nodes or deploy code to a new Puppet environment, use this command to run Puppet across all the nodes in that environment.

If you run a job on a node that has relationships outside of the target (for example, it participates in an application that includes nodes not in the job target) the job still runs only on the node in the target you specified. In such cases, the orchestrator notifies you that external relationships exist. It prints the node with relationships, and it prints the applications that might be affected. For example:

```
**WARNING** target does not contain all nodes in this application.
```

You can run jobs on three types of targets, but these targets cannot be combined:

- An application or an application instance in an environment
- A list of nodes or a single node
- A PQL nodes query

When you execute a `puppet job run` command, the orchestrator creates a new Job ID, shows you all nodes included in the job, and proceeds to run Puppet on all nodes in the appropriate order. Puppet compiles a new catalog for all nodes included in the job.

See [puppet job run command options](#) on page 484 for additional run options.

The first time you run a command, you need to authenticate. See the [orchestrator installation instructions](#) for information about setting RBAC permissions and token authorization.

If you're running this command from a managed or non-managed Windows workstation, you must specify the full path to the command. For example, `c:\Program Files\Puppet Labs\Client\bin\puppet-task run`.

Run Puppet on a list of nodes or a single node

Use a node list target for an orchestrator job when you need to run a job on a specific set of nodes that don't easily resolve to a PQL query. Use a single node or a comma-separated list of nodes.

Before you begin

Make sure you have permissions to run jobs.

Make sure you have access to the nodes you want to target.

Log into your primary server or client tools workstation and run one of the following commands:

- To run a job on a single node:

```
puppet job run --nodes <NODE NAME> <OPTIONS>
```

- To run a job on a list of nodes, use a **comma-separated** list of node names:

```
puppet job run --nodes <NODE NAME>, <NODE NAME>, <NODE NAME>, <NODE NAME>
<OPTIONS>
```

Note: Do not add spaces in the list of nodes.

- To run a job on a node list from a text file:

```
puppet job run --nodes @/path/to/file.txt
```

Note: If passing a list of nodes in the text file, put each node on a separate line.

To view the status of the Puppet job, run `puppet job show <job ID>`. To view a list of the previous 50 running and complete Puppet jobs, run `puppet job show`

Run Puppet on a PQL query

Use a PQL nodes query as a target when you want to target nodes that meet specific conditions. In this case, the orchestrator runs on a list of nodes returned from a PQL nodes query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run jobs and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- To specify the query on the command line: `puppet job run --query '<QUERY>' <OPTIONS>`
- To pass the query in a text file: `puppet job run --query @/path/to/file.txt`

The following table shows some example targets and the associated PQL queries you could run with the orchestrator.

Be sure to wrap the entire query in single quotes and use double quotes inside the query.

Target	PQL query
Single node by certname	<code>--query 'nodes { certname = "mynode" }'</code>
All nodes with "web" in certname	<code>--query 'nodes { certname ~ "web" }'</code>
All CentOS nodes	<code>--query 'inventory { facts.os.name = "CentOS" }'</code>
All CentOS nodes with httpd managed	<code>--query 'inventory { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }'</code>
All nodes with failed reports	<code>--query 'reports { latest_report? = true and status = "failed" }'</code>
All nodes matching the environment for the last received catalog	<code>--query 'nodes { catalog_environment = "production" }'</code>

To view the status of the Puppet job, run `puppet job show <job ID>`. To view a list of the previous 50 running and complete Puppet jobs, run `puppet job show`

Run Puppet on a node group

Similar to running Puppet on a list of nodes, you can run it on a node group..

Before you begin

Make sure you have permissions to run jobs.

Make sure you have access to the nodes you want to target.

1. Log into your primary server or client tools workstation.
2. Run the command: `puppet job run --node-group <node-group-id>`

Tip: Use the `/v1/groups` endpoint to retrieve a list node groups and their IDs.

Related information

[GET /v1/groups](#) on page 389

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

Run Puppet on an application or an application instance in an environment

Use applications as a job target to enforce Puppet runs in order on all nodes found in a specific application instance, or to enforce Puppet runs in order on all nodes that are found in each instance of an application.

Before you begin

Make sure you have permissions to run jobs.

Log into your primary server or client tools workstation and run one of the following commands:

- To run a job on all instances of an application: `puppet job run --application <APPLICATION> --environment <ENVIRONMENT>`
- To run a job on an instance of an application in an environment: `puppet job run --application <APPLICATION INSTANCE> --environment <ENVIRONMENT>`

Tip: You can use `-a` in place of `--application`.

To view the status of the Puppet job, run `puppet job show <job ID>`. To view a list of the previous 50 running and complete Puppet jobs, run `puppet job show`

puppet job run command options

The following are common options you can use with the run action. For a complete list of global options run `puppet job --help`.

Option	Value	Description
<code>--noop</code>	Flag, default false	Run a job on all nodes to simulate changes from a new catalog without actually enforcing a new catalog. Cannot be used in conjunction with <code>--no-noop</code> flag.
<code>--no-nopp</code>	Flag	All nodes run in enforcement mode, and a new catalog is enforced on all nodes. This flag overrides the agent <code>noop = true</code> in <code>puppet.conf</code> . Cannot be used in conjunction with <code>--noop</code> flag.

Option	Value	Description
<code>--environment, -e</code>	Environment name	<p>Overrides the environment specified in the orchestrator configuration file. The orchestrator uses this option to:</p> <ul style="list-style-type: none"> • Instruct nodes what environment to run in. If any nodes can't run in the environment, those node runs fail. A node runs in an environment if: <ul style="list-style-type: none"> • The node is included in an application in that environment. These runs may fail if the node is classified into a different environment in the PE node classifier. • The node is classified into that environment in the PE node classifier. • Load the application code used to plan run order
<code>--no-enforce-environment</code>	Flag, default false	<p>Ignores the environment set by the <code>--environment</code> flag for agent runs. When you use this flag, agents run in the environment specified by the PE Node Manager or their <code>puppet.conf</code> files.</p>
<code>--description</code>	Flag, defaults to empty	<p>Provide a description for the job, to be shown on the job list and job details pages, and returned with the <code>puppet job show</code> command.</p>
<code>--concurrency</code>	Integer	<p>Limits how many nodes can run concurrently. Default is unlimited. You can tune concurrent compile requests in the console.</p>

Post-run node status

After Puppet runs, the orchestrator returns a list of nodes and their run statuses.

Node runs can be in progress, completed, skipped, or failed.

- For a completed node run, the orchestrator prints the configuration version, the transaction ID, a summary of resource events, and a link to the full node run report in the console.
- For an in progress node run, the orchestrator prints how many seconds ago the run started.
- For a failed node run, the orchestrator prints an error message indicating why the run failed. In this case, any additional runs are skipped.

When a run fails, the orchestrator also prints any applications that were affected by the failure, as well as any applications that were affected by skipped node runs.

You can view the status of all running, completed, and failed jobs with the `puppet job show` command, or you can view them from the **Job details** page in the console.

Additionally, on the **Jobs** page in the console, you can review a list of jobs or to view the details of jobs that have previously run or are in progress.

Stop a job in progress

You can stop a job if, for example, you realize you need to reconfigure a class or push a configuration change that the job needs.

When you stop a Puppet job, jobs that are underway finish, and jobs that have not started are canceled.

To stop a job:

- In the console, go to the **Jobs** page and select the **Puppet run** tab. From the list of jobs, find the one you want and click **Stop**.
- On the command line, press **CTRL + C**.

POST /command/deploy

Run the orchestrator across all nodes in an environment.

Request format

The request body must be a JSON object using these keys:

Key	Definition
environment	The environment to deploy. This key is required.
scope	Object, required unless <code>target</code> is specified. The PuppetDB query, a list of nodes, a classifier node group id, or an application/application instance to deploy.
description	String, a description of the job.
noop	Boolean, whether to run the agent in no-op mode. The default is <code>false</code> .
no_noop	Boolean, whether to run the agent in enforcement mode. Defaults to <code>false</code> . This flag overrides <code>noop = true</code> if set in the agent's <code>puppet.conf</code> , and cannot be set to <code>true</code> at the same time as the <code>noop</code> flag.
concurrency	The maximum number of nodes to run at one time. The default is the range between 1 and up to the value set by the following parameter, which defaults to 8: <code>puppet_enterprise::profile::orchestrator::global_</code>
enforce_environment	Boolean, whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be <code>false</code> if <code>environment</code> is an empty string
debug	Boolean, whether to use the <code>--debug</code> flag on Puppet agent runs.
trace	Boolean, whether to use the <code>--trace</code> flag on Puppet agent runs.
evaltrace	Boolean, whether to use the <code>--evaltrace</code> flag on Puppet agent runs.
filetimeout	Integer, sets the <code>--filetimeout</code> flag on Puppet agent runs to the provided value.

Key	Definition
<code>http_connect_timeout</code>	Integer, sets the <code>--http_connect_timeout</code> flag on Puppet agent runs to the provided value.
<code>http_keepalive_timeout</code>	Integer, sets the <code>--http_keepalive_timeout</code> flag on Puppet agent runs to the provided value.
<code>http_read_timeout</code>	Integer, sets the <code>--http_read_timeout</code> flag on Puppet agent runs to the provided value.
<code>ordering</code>	String, sets the <code>--ordering</code> flag on Puppet agent runs to the provided value.
<code>skip_tags</code>	String, sets the <code>--skip_tags</code> flag on Puppet agent runs to the provided value.
<code>tags</code>	String, sets the <code>--tags</code> flag on Puppet agent runs to the provided value.
<code>use_cached_catalog</code>	Boolean, whether to use the <code>--use_cached_catalog</code> flag on Puppet agent runs.
<code>usecacheonfailure</code>	Boolean, whether to use the <code>--usecacheonfailure</code> flag on Puppet agent runs.
<code>userdata</code>	An object of arbitrary key/value data supplied to the job.

For example, to deploy the `node1.example.com` environment in no-op mode, the following request is valid:

```
{
  "environment" : "",
  "enforce_environment": false,
  "noop" : true,
  "scope" : {
    "nodes" : ["node1.example.com"]
  },
  "userdata": {
    "servicenow_ticket": "INC0011211"
  }
}
```

Scope

Scope is a JSON object containing exactly one of these keys:

Key	Definition
<code>application</code>	The name of an application or application instance to deploy. If an application type is specified, all instances of that application are deployed.
<code>nodes</code>	A list of node names to target.
<code>query</code>	A PuppetDB or PQL query to use to discover nodes. The target is built from <code>certname</code> values collected at the top level of the query.
<code>node_group</code>	A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group.

To deploy an application instance in the production environment:

```
{
  "environment" : "production",
  "scope" : {
    "application" : "Wordpress_app[demo]"
  }
}
```

To deploy a list of nodes:

```
{
  "environment" : "production",
  "scope" : {
    "nodes" : [ "node1.example.com", "node2.example.com" ]
  }
}
```

To deploy a list of nodes with the certname value matching a regex:

```
{
  "environment" : "production",
  "scope" : {
    "query" : [ "from", "nodes", [ "~", "certname", ".*" ] ]
  }
}
```

To deploy to the nodes defined by the "All Nodes" node group:

```
{
  "environment" : "production",
  "scope" : {
    "node_group" : "000000000-0000-4000-8000-0000000000000"
  }
}
```

Response format

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response is a JSON object containing a link to retrieve information about the status of the job and uses any one of these keys:

Key	Definition
id	An absolute URL that links to the newly created job.
name	The name of the newly created job.

For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234"
    "name" : "1234"
  }
}
```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
<code>puppetlabs.orchestrator/unknown-environment</code>	If the environment does not exist, the server returns a 404 response.
<code>puppetlabs.orchestrator/empty-environment</code>	If the environment requested contains no applications or no nodes, the server returns a 400 response.
<code>puppetlabs.orchestrator/empty-target</code>	If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.
<code>puppetlabs.orchestrator/dependency-cycle</code>	If the application code contains a cycle, the server returns a 400 response.
<code>puppetlabs.orchestrator/puppetdb-error</code>	If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.
<code>puppetlabs.orchestrator/query-error</code>	If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.

Related information

[Puppet orchestrator API: forming requests](#) on page 549

Instructions on interacting with this API.

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Tasks in PE

Tasks are ad-hoc actions you can execute on a target and run from the command line or the console.

A *task* is a single action that you execute on target machines. With tasks, you can troubleshoot and deploy changes to individual or multiple systems in your infrastructure.

You can run tasks from your tool of choice: the console, the orchestrator command line interface (CLI), or the orchestrator API [/command/task endpoint](#).

When you run a task, you can run it immediately, schedule it to run later, or schedule it to run at a recurring frequency - hourly, daily, weekly, every 2 weeks, or every four weeks. After you launch a task, you can check on the status or view the output later with the console or CLI.

Note: If you are running multiple tasks, make sure your concurrency limit for tasks and bolt-server can accommodate your needs. See [Configure the orchestrator and pe-orchestration-services](#) on page 173 to adjust these settings.

- [Installing tasks](#) on page 489

Puppet Enterprise comes with some pre-installed tasks, but you must install or write other tasks you want to use.

- [Running tasks in PE](#) on page 490

Use the orchestrator to set up jobs in the console or on the command line and run Bolt tasks across systems in your infrastructure.

- [Writing tasks](#) on page 501

Bolt tasks are similar to scripts, but they are kept in modules and can have metadata. This allows you to reuse and share them.

Installing tasks

Puppet Enterprise comes with some pre-installed tasks, but you must install or write other tasks you want to use.

Tasks are packaged in Puppet modules and can be installed from the [Forge](#), then managed with a Puppetfile and Code Manager.

To install a new task, select the desired install method under **Start using this module** on the module's page in the Forge and follow the instructions.

PE comes with the following tasks already installed:

- `package` - Inspect, install, upgrade, and manage packages.
- `service` - Start, stop, restart, and check the status of a service.
- `facter_task` - Inspect the value of system facts.
- `puppet_conf` - Inspect Puppet agent configuration settings.

Related information

[Managing environment content with a Puppetfile](#) on page 620

A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Running tasks in PE

Use the orchestrator to set up jobs in the console or on the command line and run Bolt tasks across systems in your infrastructure.

Running a task does not update your Puppet configuration. If you run a task that changes the state of a resource that Puppet is managing, a subsequent Puppet run changes the state of that resource back to what is defined in your Puppet configuration. For example, if you use a task to update the version of a managed package, the version of that package is reset to whatever is specified in a manifest on the next Puppet run.

Note: If you have set up compilers and you want to use tasks, you must either set `master_uris` or your `server_list` on agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.

- [Running tasks from the console](#) on page 490

Run ad-hoc tasks on target machines to upgrade packages, restart services, or perform any other type of single-action executions on your nodes.

- [Running tasks from the command line](#) on page 497

Use the `puppet task run` command to run tasks on agent nodes.

- [Stop a task in progress](#) on page 500

You can stop a task in progress if, for example, you realize you need to adjust your PQL query or edit the parameters the task run needs.

- [Inspecting tasks](#) on page 500

View the tasks that you have installed and have permission to run, as well as the documentation for those tasks.

Related information

[Configure compilers](#) on page 136

Compilers must be configured to appropriately route communication between your primary server and agent nodes.

Running tasks from the console

Run ad-hoc tasks on target machines to upgrade packages, restart services, or perform any other type of single-action executions on your nodes.

When you set up a job to run a task from the console, the orchestrator creates a job ID to track the job, shows you all nodes included in the job, and runs the tasks on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

There are three ways to specify the job target (the nodes you want to run tasks on):

- A static node list
- A Puppet Query Language (PQL) query
- A node group

You can't combine these methods, and if you switch from one to the other, the target list clears and starts over. In other words, if you create a target list by using a node list, switching to a PQL query clears the node list. You can do a one-time conversion of a PQL query to a static node list if you want to add or remove nodes from the query results.

Run a task on a node list

Create a list of target nodes when you need to run a task on a specific set of nodes that isn't easily defined by a PQL query.

Before you begin

Install the tasks you want to use.

Make sure you have permission to run the tasks.

Make sure you have access to the nodes you want to target.

Tip: You can add network devices to a node list when you have installed modules for device transports in your production environment. You can find such modules in [Puppet Forge](#).

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to `production`.
4. In the **Task** field, select a task to run, for example `service`.
Note: If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.
5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

Important: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

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

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
8. From the list of target types, select **Node list**.
9. Create the node list.
 - a) Expand the **Inventory nodes** target.
 - b) Enter the name of the node you want to find and click **Search**.

Note: The search does not handle regular expressions but does support partial matches.

- c) From the list of results, select the nodes that you want to add to your list. They are added to a table below.
- d) Repeat the search to add other nodes. You can select nodes from multiple searches to create the node list.

Tip: To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

Run a task over SSH

Use the SSH protocol to run tasks on target nodes that do not have the Puppet agent installed.

Before you begin

Install the tasks you want to use.

Make sure you have permission to run the tasks on all nodes.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to `production`.
4. In the **Task** field, select a task to run, for example `service`.

Note: If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

Important: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

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

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
8. From the list of target types, select **Node list**.
9. Create the node list.

- a) Expand the **SSH nodes** target.

Note: This target is available only for tasks permitted to run on all nodes.

- b) Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
- c) Optional: Select additional target options.
For example, to add a target port number, select **Target port** from the drop-down list, enter the number, and click **Add**.
- d) Click **Add nodes**. They are added to a table below.
- e) Repeat these steps to add other nodes. You can add SSH nodes with different credentials to create the node list.

Tip: To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

Run a task over WinRM

Use the Windows Remote Management (WinRM) to run tasks on target nodes that do not have the Puppet agent installed.

Before you begin

Install the tasks you want to use.

Make sure you have permission to run the tasks on all nodes.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to `production`.
4. In the **Task** field, select a task to run, for example `service`.

Note: If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

Important: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

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

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
8. From the list of target types, select **Node list**.
9. Create the node list.
 - a) Expand the **WinRM nodes** target.

Note: This target is available only for tasks permitted to run on all nodes.

- b) Enter the target host names and the credentials required to access them.
- c) Optional: Select additional target options.
For example, to add a target port number, select **Target Port** from the drop-down list, enter the number, and click **Add**.
- d) Click **Add nodes**. They are added to a table below.
- e) Repeat these steps to add other nodes. You can add nodes with different credentials to create the node list.

Tip: To remove a node from the table, select the checkbox next to it and click **Remove selected**.

10. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

Run a task on a PQL query

Create a PQL query to run tasks on nodes that meet specific conditions.

Before you begin

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run tasks and PQL queries.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to `production`.
4. In the **Task** field, select a task to run, for example `service`.

Note: If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

Important: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

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

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
8. From the list of target types, select **PQL query**.

9. Specify a target by doing one of the following:

- Enter a query that selects the target you want. See the [Puppet Query Language \(PQL\) reference](#) for more information.
- Click **Common queries**. Select one of the queries and replace the defaults in the braces (`{ }`) with values that specify the target you want.

Target	PQL query
All nodes	<code>nodes[certname] { }</code>
Nodes with a specific resource (example: httpd)	<code>resources[certname] { type = "Service" and title = "httpd" }</code>
Nodes with a specific fact and value (example: OS name is CentOS)	<code>inventory[certname] { facts.os.name = "<OS>" }</code>
Nodes with a specific report status (example: last run failed)	<code>reports[certname] { latest_report_status = "failed" }</code>
Nodes with a specific class (example: Apache)	<code>resources[certname] { type = "Class" and title = "Apache" }</code>
Nodes assigned to a specific environment (example: production)	<code>nodes[certname] { catalog_environment = "production" }</code>
Nodes with a specific version of a resource type (example: OpenSSL is v1.1.0e)	<code>resources[certname] { type = "Package" and title="openssl" and parameters.ensure = "1.0.1e-51.el7_2.7" }</code>
Nodes with a specific resource and operating system (example: httpd and CentOS)	<code>inventory[certname] { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }</code>

10. Click **Submit query** and click **Refresh** to update the node results.

11. If you change or edit the query after it runs, click **Submit query** again.

12. Optional: To convert the PQL query target results to a node list, for use as a node list target, click **Convert query to static node list**.

Note: If you select this option, the job target becomes a node list. You can add or remove nodes from the node list before running the job, but you cannot edit the query.

13. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

Important: When you run this job, the PQL query runs again, and the job might run on a different set of nodes than what is currently displayed. If you want the job to run only on the list as displayed, convert the query to a static node list before you run the job.

Add custom PQL queries to the console

Add your own PQL queries to the console and quickly access them when running jobs.

1. On the primary sever, as root, copy the `custom_pql_queries.json.example` file and remove the `example` suffix.

```
cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

2. Edit the file contents to include your own PQL queries or remove any existing queries.
3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop down options when running jobs.

Run a task on a node group

Similar to running a task on a list of nodes that you create in the console, you can run a task on a node group.

Before you begin

Install the tasks you want to use.

Make sure you have permission to run the tasks.

Make sure you have access to the nodes you want to target.

Note: If you do not have permissions to view a node group, or the node group doesn't have any matching nodes, that node group won't be listed as an option for viewing. In addition, a node group does not appear if no rules have been specified for it.

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to `production`.
4. In the **Task** field, select a task to run, for example `service`.

Note: If the tasks you expect are not available, you either have no tasks installed, or you don't have the correct permissions to run them.

5. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
6. Under **Task parameters**, add optional parameters and enter values for the optional and required parameters on the list.

Important: You must click **Add parameter** for **each optional** parameter-value pair you add to the task.

To view information about required and optional parameters for the task, select **view task metadata** below the **Task** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Tasks with default values run using the default value unless another value is specified.

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

7. Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
8. From the list of target types, select **Node group**.
9. In the **Choose a node group** box, type or select a node group, and click **Select**.
10. Click **Run task** or **Schedule job**.

Your task run appears on the **Tasks** page. To rerun the task, click **Run again** and choose to rerun the task on all nodes or only on the nodes that failed during the initial run.

Tip: Filter run results by task name to find specific task runs.

Running tasks from the command line

Use the `puppet task run` command to run tasks on agent nodes.

Use the `puppet task` tool and the relevant module to make changes arbitrarily, rather than through a Puppet configuration change. For example, to inspect a package or quickly stop a particular service.

You can run tasks on a single node, on nodes identified in a static list, on nodes retrieved by a PQL query, or on nodes in a node group.

Use the orchestrator command `puppet task` to trigger task runs.

The first time you run a command, you need to authenticate. See the [orchestrator installation instructions](#) for information about setting RBAC permissions and token authorization.

Run a task on a list of nodes or a single node

Use a node list target when you need to run a job on a set of nodes that doesn't easily resolve to a PQL query. Use a single node or a comma-separated list of nodes.

Before you begin

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run tasks and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- To run a task job on a single node: `puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --nodes <NODE NAME> <OPTIONS>`
- To run a task job on a list of nodes, use a comma-separated list of node names: `puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --nodes <NODE NAME>,<NODE NAME>,<NODE NAME> <OPTIONS>`

Note: Do not add spaces in the list of nodes.

- To run a task job on a node list from a text file: `puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --nodes @/path/to/file.txt`

Note: In the text file, put each node on a separate line.

For example, to run the service task with two required parameters, on three specific hosts:

```
puppet task run service action=status service=nginx --nodes
host1,host2,host3
```

Tip: Use `puppet task show <TASK NAME>` to see a list of available parameters for a task. Not all tasks require parameters.

Refer to the `puppet task` command options to see how to pass parameters with the `--params` flag.

Run a task on a PQL query

Create a PQL query to run tasks on nodes that meet specific conditions.

Before you begin

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

Make sure you have permissions to run tasks and PQL queries.

Log into your primary server or client tools workstation and run one of the following commands:

- To specify the query on the command line: `puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --query '<QUERY>' <OPTIONS>`

- To pass the query in a text file: `puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --query @/path/to/file.txt`

For example, to run the service task with two required parameters, on nodes with "web" in their certname:

```
puppet task run service action=status service=nginx --query 'nodes
{ certname ~ "web" }'
```

Tip: Use `puppet task show <TASK NAME>` to see a list of available parameters for a task. Not all tasks require parameters.

Refer to the `puppet task` command options to see how to pass parameters with the `--params` flag.

The following table shows some example targets and the associated PQL queries you could run with the orchestrator.

Be sure to wrap the entire query in single quotes and use double quotes inside the query.

Target	PQL query
Single node by certname	<code>--query 'nodes { certname = "mynode" }'</code>
All nodes with "web" in certname	<code>--query 'nodes { certname ~ "web" }'</code>
All CentOS nodes	<code>--query 'inventory { facts.os.name = "CentOS" }'</code>
All CentOS nodes with httpd managed	<code>--query 'inventory { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }'</code>
All nodes with failed reports	<code>--query 'reports { latest_report? = true and status = "failed" }'</code>
All nodes matching the environment for the last received catalog	<code>--query 'nodes { catalog_environment = "production" }'</code>

Tip: You can use `-q` in place of `--query`.

Run a task on a node group

Similar to running a task on a list of nodes, you can run a task on a node group.

Before you begin

Install the tasks you want to use.

Make sure you have access to the nodes you want to target.

1. Log into your primary server or client tools workstation.
2. Run the command: `puppet task run <TASK NAME> --node-group <node-group-id>`

Tip: Use the `/v1/groups` endpoint to retrieve a list node groups and their IDs.

Related information

[GET /v1/groups](#) on page 389

Use the `/v1/groups` endpoint to retrieve a list of all node groups in the node classifier.

puppet task run command options

The following are common options you can use with the `task` action. For a complete list of global options run `puppet task --help`.

Option	Value	Description
<code>--noop</code>	Flag, default false	Run a task to simulate changes without actually enforcing the changes.
<code>--params</code>	String	Specify a JSON object that includes the parameters, or specify the path to a JSON file containing the parameters, prefaced with <code>@</code> , for example, <code>@/path/to/file.json</code> . Do not use this flag if specifying parameter-value pairs inline; see more information below.
<code>--environment, -e</code>	Environment name	Use tasks installed in the specified environment.
<code>--description</code>	Flag, defaults to empty	Provide a description for the job, to be shown on the job list and job details pages, and returned with the <code>puppet job show</code> command.

You can pass parameters into the task one of two ways:

- Inline, using the `<PARAMETER>=<VALUE>` syntax:

```
puppet task run <TASK NAME> <PARAMETER>=<VALUE>, <PARAMETER>=<VALUE> --
nodes <LIST OF NODES>
puppet task run my_task action=status service=my_service timeout=8 --nodes
host1,host2,host3
```

- With the `--params` option, as a JSON object or reference to a JSON file:

```
puppet task run <TASK NAME> --params '<JSON OBJECT>' --nodes <LIST OF
NODES>
puppet task run my_task --params '{ "action":"status",
  "service":"my_service", "timeout":8 }' --nodes host1,host2,host3
puppet task run my_task --params @/path/to/file.json --nodes
host1,host2,host3
```

You can't combine these two ways of passing in parameters; choose either inline or `--params`. If you use the inline way, parameter types other than string, integer, double, and Boolean will be interpreted as strings. Use the `--params` method if you want them read as their original type.

Reviewing task job output

The output the orchestrator returns depends on the type of task you run. Output is either standard output (STDOUT) or structured output. At minimum, the orchestrator prints a new job ID and the number of nodes in the task.

The following example shows a task to check the status of the Puppet service running on a list of nodes derived from a PQL query.

```
[example@orch-master ~]$ puppet task run service service=puppet
  action=status -q 'nodes {certname ~ "br"}' --environment=production
Starting job ...
New job ID: 2029
```

```

Nodes: 8

Started on bronze-11 ...
Started on bronze-8 ...
Started on bronze-3 ...
Started on bronze-6 ...
Started on bronze-2 ...
Started on bronze-5 ...
Started on bronze-7 ...
Started on bronze-10 ...
Finished on node bronze-11
  status : running
  enabled : true
Finished on node bronze-3
  status : running
  enabled : true
Finished on node bronze-8
  status : running
  enabled : true
Finished on node bronze-7
  status : running
  enabled : true
Finished on node bronze-2
  status : running
  enabled : true
Finished on node bronze-6
  status : running
  enabled : true
Finished on node bronze-5
  status : running
  enabled : true
Finished on node bronze-10
  status : running
  enabled : true

Job completed. 8/8 nodes succeeded.
Duration: 1 sec

```

Tip: To view the status of all running, completed, and failed jobs run the `puppet job show` command, or view them from the **Job details** page in the console.

Stop a task in progress

You can stop a task in progress if, for example, you realize you need to adjust your PQL query or edit the parameters the task run needs.

When you stop a task run, runs that are underway still finish but no new tasks start on the node until you run again.

To stop a task, do one of the following:

- In the console, on the **Tasks** page, find the task run you want to stop and click **Stop job**.
- On the command line, press **CTRL + C**.

Inspecting tasks

View the tasks that you have installed and have permission to run, as well as the documentation for those tasks.

Log into your primary server or client tools workstation and run one of the following commands:

- To check the documentation for a specific task: `puppet task show <TASK>`. The command returns the following:
 - The command format for running the task
 - Any parameters available to use with the task

- To view a list of your permitted tasks: `puppet task show`
- To view a list of all installed tasks pass the `--all` flag: `puppet task show --all`

Writing tasks

Bolt tasks are similar to scripts, but they are kept in modules and can have metadata. This allows you to reuse and share them.

You can write tasks in any programming language the target nodes run, such as Bash, PowerShell, or Python. A task can even be a compiled binary that runs on the target. Place your task in the `./tasks` directory of a module and add a metadata file to describe parameters and configure task behavior.

For a task to run on remote *nix systems, it must include a shebang (`#!`) line at the top of the file to specify the interpreter.

For example, the Puppet `mysql::sql` task is written in Ruby and provides the path to the Ruby interpreter. This example also accepts several parameters as JSON on `stdin` and returns an error.

```
#!/opt/puppetlabs/puppet/bin/ruby
require 'json'
require 'open3'
require 'puppet'

def get(sql, database, user, password)
  cmd = ['mysql', '-e', "#{sql} "]
  cmd << "--database=#{database}" unless database.nil?
  cmd << "--user=#{user}" unless user.nil?
  cmd << "--password=#{password}" unless password.nil?
  stdout, stderr, status = Open3.capture3(*cmd) # rubocop:disable Lint/
  UselessAssignment
  raise Puppet::Error, _("#{stderr: ' %{stderr}')} % { stderr: stderr }") if
  status != 0
  { status: stdout.strip }
end

params = JSON.parse(STDIN.read)
database = params['database']
user = params['user']
password = params['password']
sql = params['sql']

begin
  result = get(sql, database, user, password)
  puts result.to_json
  exit 0
rescue Puppet::Error => e
  puts({ status: 'failure', error: e.message }.to_json)
  exit 1
end
```

Related information

[Task compatibility](#) on page 14

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

Secure coding practices for tasks

Use secure coding practices when you write tasks and help protect your system.

Note: The information in this topic covers basic coding practices for writing secure tasks. It is not an exhaustive list.

One of the methods attackers use to gain access to your systems is remote code execution, where by running an allowed script they gain access to other parts of the system and can make arbitrary changes. Because Bolt executes

scripts across your infrastructure, it is important to be aware of certain vulnerabilities, and to code tasks in a way that guards against remote code execution.

Adding task metadata that validates input is one way to reduce vulnerability. When you require an enumerated (enum) or other non-string types, you prevent improper data from being entered. An arbitrary string parameter does not have this assurance.

For example, if your task has a parameter that selects from several operational modes that are passed to a shell command, instead of

```
String $mode = 'file'
```

use

```
Enum[file,directory,link,socket] $mode = file
```

If your task has a parameter that identifies a file on disk, ensure that a user can't specify a relative path that takes them into areas where they shouldn't be. Reject file names that have slashes.

Instead of

```
String $path
```

use

```
Pattern[/\A[^\s\\]*\z/] $path
```

In addition to these task restrictions, different scripting languages each have their own ways to validate user input.

PowerShell

In PowerShell, code injection exploits calls that specifically evaluate code. Do not call `Invoke-Expression` or `Add-Type` with user input. These commands evaluate strings as C# code.

Reading sensitive files or overwriting critical files can be less obvious. If you plan to allow users to specify a file name or path, use `Resolve-Path` to verify that the path doesn't go outside the locations you expect the task to access. Use `Split-Path -Parent $path` to check that the resolved path has the desired path as a parent.

For more information, see [PowerShell Scripting](#) and [Powershell's Security Guiding Principles](#).

Bash

In Bash and other command shells, shell command injection takes advantage of poor shell implementations. Put quotations marks around arguments to prevent the vulnerable shells from evaluating them.

Because the `eval` command evaluates all arguments with string substitution, avoid using it with user input; however you can use `eval` with sufficient quoting to prevent substituted variables from being executed.

Instead of

```
eval "echo $input"
```

use

```
eval "echo '$input' "
```

These are operating system-specific tools to validate file paths: `realpath` or `readlink -f`.

Python

In Python malicious code can be introduced through commands like `eval`, `exec`, `os.system`, `os.popen`, and `subprocess.call` with `shell=True`. Use `subprocess.call` with `shell=False` when you include user input in a command or escape variables.

Instead of

```
os.system('echo '+input)
```

use

```
subprocess.check_output(['echo', input])
```

Resolve file paths with `os.realpath` and confirm them to be within another path by looping over `os.path.dirname` and comparing to the desired path.

For more information on the vulnerabilities of Python or how to escape variables, see Kevin London's blog post on [Dangerous Python Functions](#).

Ruby

In Ruby, command injection is introduced through commands like `eval`, `exec`, `system`, backtick (```) or `%x()` execution, or the `Open3` module. You can safely call these functions with user input by passing the input as additional arguments instead of a single string.

Instead of

```
system("echo #{flag1} #{flag2}")
```

use

```
system('echo', flag1, flag2)
```

Resolve file paths with `Pathname#realpath`, and confirm them to be within another path by looping over `Pathname#parent` and comparing to the desired path.

For more information on securely passing user input, see the blog post [Stop using backtick to run shell command in Ruby](#).

Naming tasks

Task names are named based on the filename of the task, the name of the module, and the path to the task within the module.

You can write tasks in any language that runs on the target nodes. Give task files the extension for the language they are written in (such as `.rb` for Ruby), and place them in the top level of your module's `./tasks` directory.

Task names are composed of one or two name segments, indicating:

- The name of the module where the task is located.
- The name of the task file, without the extension.

For example, the `puppetlabs-mysql` module has the `sql` task in `./mysql/tasks/sql.rb`, so the task name is `mysql::sql`. This name is how you refer to the task when you run tasks.

The task filename `init` is special: the task it defines is referenced using the module name only. For example, in the `puppetlabs-service` module, the task defined in `init.rb` is the `service` task.

Each task or plan name segment must begin with a lowercase letter and:

- Must start with a lowercase letter.
- Can include digits.

- Can include underscores.
- Namespace segments must match the following regular expression `\A[a-z][a-z0-9_]*\Z`
- The file extension must not use the reserved extensions `.md` or `.json`.

Single-platform tasks

A task can consist of a single executable with or without a corresponding metadata file. For instance, `./mysql/tasks/sql.rb` and `./mysql/tasks/sql.json`. In this case, no other `./mysql/tasks/sql.*` files can exist.

Cross-platform tasks

A task can have multiple implementations, with metadata that explains when to use each one. A primary use case for this is to support different implementations for different target platforms, referred to as *cross-platform tasks*.

A task can also have multiple implementations, with metadata that explains when to use each one. A primary use case for this is to support different implementations for different target platforms, referred to as *cross-platform tasks*. For instance, consider a module with the following files:

```
- tasks
  - sql_linux.sh
  - sql_linux.json
  - sql_windows.ps1
  - sql_windows.json
  - sql.json
```

This task has two executables (`sql_linux.sh` and `sql_windows.ps1`), each with an implementation metadata file and a task metadata file. The executables have distinct names and are compatible with older task runners such as Puppet Enterprise 2018.1 and earlier. Each implementation has its own metadata which documents how to use the implementation directly or marks it as private to hide it from UI lists.

An implementation metadata example:

```
{
  "name": "SQL Linux",
  "description": "A task to perform sql operations on linux targets",
  "private": true
}
```

The task metadata file contains an implementations section:

```
{
  "implementations": [
    { "name": "sql_linux.sh", "requirements": ["shell"] },
    { "name": "sql_windows.ps1", "requirements": ["powershell"] }
  ]
}
```

Each implementation has a name and a list of *requirements*. The requirements are the set of *features* which must be available on the target in order for that implementation to be used. In this case, the `sql_linux.sh` implementation requires the `shell` feature, and the `sql_windows.ps1` implementation requires the PowerShell feature.

The set of features available on the target is determined by the task runner. You can specify additional features for a target via `set_feature` or by adding features in the inventory. The task runner chooses the *first* implementation whose requirements are satisfied.

The following features are defined by default:

- `puppet-agent`: Present if the target has the Puppet agent package installed. This feature is automatically added to hosts with the name `localhost`.
- `shell`: Present if the target has a posix shell.

- `powershell`: Present if the target has PowerShell.

Sharing executables

Multiple task implementations can refer to the same executable file.

Executables can access the `_task` metaparameter, which contains the task name. For example, the following creates the tasks `service::stop` and `service::start`, which live in the executable but appear as two separate tasks.

```
myservice/tasks/init.rb
```

```
#!/usr/bin/env ruby
require 'json'

params = JSON.parse(STDIN.read)
action = params['action'] || params['_task']
if ['start', 'stop'].include?(action)
  `systemctl #{params['_task']} #{params['service']}`
end
```

```
myservice/tasks/start.json
```

```
{
  "description": "Start a service",
  "parameters": {
    "service": {
      "type": "String",
      "description": "The service to start"
    }
  },
  "implementations": [
    { "name": "init.rb" }
  ]
}
```

```
myservice/tasks/stop.json
```

```
{
  "description": "Stop a service",
  "parameters": {
    "service": {
      "type": "String",
      "description": "The service to stop"
    }
  },
  "implementations": [
    { "name": "init.rb" }
  ]
}
```

Sharing task code

Multiple tasks can share common files between them. Tasks can additionally pull library code from other modules.

To create a task that includes additional files pulled from modules, include the `files` property in your metadata as an array of paths. A path consists of:

- the module name

- one of the following directories within the module:
 - `files` — Most helper files. This prevents the file from being treated as a task or added to the Puppet Ruby loadpath.
 - `tasks` — Helper files that can be called as tasks on their own.
 - `lib` — Ruby code that might be reused by types, providers, or Puppet functions.
- the remaining path to a file or directory; directories must include a trailing slash /

All path separators must be forward slashes. An example would be `stdlib/lib/puppet/`.

The `files` property can be included both as a top-level metadata property, and as a property of an implementation, for example:

```
{
  "implementations": [
    { "name": "sql_linux.sh", "requirements": ["shell"], "files": ["mymodule/
files/lib.sh"] },
    { "name": "sql_windows.ps1", "requirements": ["powershell"], "files":
["mymodule/files/lib.ps1"] }
  ],
  "files": ["emoji/files/emojis/"]
}
```

When a task includes the `files` property, all files listed in the top-level property and in the specific implementation chosen for a target are copied to a temporary directory on that target. The directory structure of the specified files is preserved such that paths specified with the `files` metadata option are available to tasks prefixed with `_installdir`. The task executable itself is located in its module location under the `_installdir` as well, so other files can be found at `../../mymodule/files/` relative to the task executable's location.

For example, you can create a task and metadata in a module at `~/puppetlabs/bolt/site-modules/mymodule/tasks/task.{json,rb}`.

Metadata

```
{
  "files": ["multi_task/files/rb_helper.rb"]
}
```

File resource

`multi_task/files/rb_helper.rb`

```
def useful_ruby
  { helper: "ruby" }
end
```

Task

```
#!/usr/bin/env ruby
require 'json'

params = JSON.parse(STDIN.read)
require_relative File.join(params['_installdir'], 'multi_task', 'files',
'rb_helper.rb')
# Alternatively use relative path
# require_relative File.join(__dir__, '..', '..', 'multi_task', 'files',
'rb_helper.rb')
puts useful_ruby.to_json
```

Output

```
Started on localhost...
```

```
Finished on localhost:
{
  "helper": "ruby"
}
Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

Task helpers

To help with writing tasks, Bolt includes [python_task_helper](#) and [ruby_task_helper](#). It also makes a useful demonstration of including code from another module.

Python example

Create task and metadata in a module at `~/.puppetlabs/bolt/site-modules/mymodule/tasks/task.{json,py}`.

Metadata

```
{
  "files": ["python_task_helper/files/task_helper.py"],
  "input_method": "stdin"
}
```

Task

```
#!/usr/bin/env python
import os, sys
sys.path.append(os.path.join(os.path.dirname(__file__), '..', '..',
                             'python_task_helper', 'files'))
from task_helper import TaskHelper

class MyTask(TaskHelper):
    def task(self, args):
        return {'greeting': 'Hi, my name is '+args['name']}

if __name__ == '__main__':
    MyTask().run()
```

Output

```
$ bolt task run mymodule::task -n localhost name='Julia'
Started on localhost...
Finished on localhost:
{
  "greeting": "Hi, my name is Julia"
}
Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

Ruby example

Create task and metadata in a new module at `~/.puppetlabs/bolt/site-modules/mymodule/tasks/mytask.{json,rb}`.

Metadata

```
{
  "files": ["ruby_task_helper/files/task_helper.rb"],
  "input_method": "stdin"
}
```

```
}
```

Task

```
#!/usr/bin/env ruby
require_relative '../..../ruby_task_helper/files/task_helper.rb'

class MyTask < TaskHelper
  def task(name: nil, **kwargs)
    { greeting: "Hi, my name is #{name}" }
  end
end

MyTask.run if __FILE__ == $0
```

Output

```
$ bolt task run mymodule::mytask -n localhost name="Robert"); DROP TABLE
Students;--"
Started on localhost...
Finished on localhost:
{
  "greeting": "Hi, my name is Robert"); DROP TABLE Students;--"
}
Successful on 1 node: localhost
Ran on 1 node in 0.12 seconds
```

Writing remote tasks

Some targets are hard or impossible to execute tasks on directly. In these cases, you can write a task that runs on a proxy target and remotely interacts with the real target.

For example, a network device might have a limited shell environment or a cloud service might be driven only by HTTP APIs. By writing a remote task, Bolt allows you to specify connection information for remote targets in their inventory file and injects them into the `_target` metaparam.

This example shows how to write a task that posts messages to Slack and reads connection information from `inventory.yaml`:

```
#!/usr/bin/env ruby
# modules/slack/tasks/message.rb

require 'json'
require 'net/http'

params = JSON.parse(STDIN.read)
# the slack API token is passed in from inventory
token = params['_target']['token']

uri = URI('https://slack.com/api/chat.postMessage')
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true

req = Net::HTTP::Post.new(uri, 'Content-type' => 'application/json')
req['Authorization'] = "Bearer #{params['_target']['token']}"
req.body = { channel: params['channel'], text: params['message'] }.to_json

resp = http.request(req)

puts resp.body
```

To prevent accidentally running a normal task on a remote target and breaking its configuration, Bolt won't run a task on a remote target unless its metadata defines it as remote:

```
{
  "remote": true
}
```

Add Slack as a remote target in your inventory file:

```
---
nodes:
  - name: my_slack
    config:
      transport: remote
      remote:
        token: <SLACK_API_TOKEN>
```

Finally, make `my_slack` a target that can run the `slack::message`:

```
bolt task run slack::message --nodes my_slack message="hello" channel=<slack
channel id>
```

Defining parameters in tasks

Allow your task to accept parameters as either environment variables or as a JSON hash on standard input.

Tasks can receive input as either environment variables, a JSON hash on standard input, or as PowerShell arguments. By default, the task runner submits parameters as both environment variables and as JSON on `stdin`.

If your task needs to receive parameters only in a certain way, such as `stdin` only, you can set the input method in your task metadata. For Windows tasks, it's usually better to use tasks written in PowerShell. See the related topic about task metadata for information about setting the input method.

Environment variables are the easiest way to implement parameters, and they work well for simple JSON types such as strings and numbers. For arrays and hashes, use structured input instead because parameters with undefined values (`nil`, `undef`) passed as environment variables have the `String` value `null`. For more information, see [Structured input and output](#) on page 511.

To add parameters to your task as environment variables, pass the argument prefixed with the Puppet task prefix `PT_`.

For example, to add a `message` parameter to your task, read it from the environment in task code as `PT_message`. When the user runs the task, they can specify the value for the parameter on the command line as `message=hello`, and the task runner submits the value `hello` to the `PT_message` variable.

```
#!/usr/bin/env bash
echo your message is $PT_message
```

Defining parameters in Windows

For Windows tasks, you can pass parameters as environment variables, but it's easier to write your task in PowerShell and use named arguments. By default tasks with a `.ps1` extension use PowerShell standard argument handling.

For example, this PowerShell task takes a process name as an argument and returns information about the process. If no parameter is passed by the user, the task returns all of the processes.

```
[CmdletBinding()]
Param(
  [Parameter(Mandatory = $False)]
  [String]
  $Name
```

```

)

if ($Name -eq $null -or $Name -eq "") {
  Get-Process
} else {
  $processes = Get-Process -Name $Name
  $result = @(
    foreach ($process in $processes) {
      $result += @{ "Name" = $process.ProcessName;
                    "CPU" = $process.CPU;
                    "Memory" = $process.WorkingSet;
                    "Path" = $process.Path;
                    "Id" = $process.Id }
    }
  )
  if ($result.Count -eq 1) {
    ConvertTo-Json -InputObject $result[0] -Compress
  } elseif ($result.Count -gt 1) {
    ConvertTo-Json -InputObject @{"_items" = $result} -Compress
  }
}

```

To pass parameters in your task as environment variables (PT_parameter), you must set `input_method` in your task metadata to `environment`. To run Ruby tasks on Windows, the Puppet agent must be installed on the target nodes.

Returning errors in tasks

To return a detailed error message if your task fails, include an `Error` object in the task's result.

When a task exits non-zero, the task runner checks for an error key (`_error`). If one is not present, the task runner generates a generic error and adds it to the result. If there is no text on `stdout` but text is present on `stderr`, the `stderr` text is included in the message.

```

{
  "_error": {
    "msg": "Task exited 1:\nSomething on stderr",
    "kind": "puppetlabs.tasks/task-error",
    "details": { "exitcode": 1 }
  }
}

```

An error object includes the following keys:

msg

A human readable string that appears in the UI.

kind

A standard string for machines to handle. You can share kinds between your modules or namespace kinds per module.

details

An object of structured data about the tasks.

Tasks can provide more details about the failure by including their own error object in the result at `_error`.

```

#!/opt/puppetlabs/puppet/bin/ruby

require 'json'

begin
  params = JSON.parse(STDIN.read)
  result = {}
  result['result'] = params['dividend'] / params['divisor']
rescue ZeroDivisionError

```

```

    result[:_error] = { msg: "Cannot divide by zero",
                        # namespace the error to this module
                        kind: "puppetlabs-example_modules/dividebyzero",
                        details: { divisor: divisor },
                      }
  rescue Exception => e
    result[:_error] = { msg: e.message,
                        kind: "puppetlabs-example_modules/unknown",
                        details: { class: e.class.to_s },
                      }
  end

  puts result.to_json

```

Structured input and output

If you have a task that has many options, returns a lot of information, or is part of a task plan, consider using structured input and output with your task.

The task API is based on JSON. Task parameters are encoded in JSON, and the task runner attempts to parse the output of the tasks as a JSON object.

The task runner can inject keys into that object, prefixed with `_`. If the task does not return a JSON object, the task runner creates one and places the output in an `_output` key.

Structured input

For complex input, such as hashes and arrays, you can accept structured JSON in your task.

By default, the task runner passes task parameters as both environment variables and as a single JSON object on stdin. The JSON input allows the task to accept complex data structures.

To accept parameters as JSON on stdin, set the `params` key to accept JSON on stdin.

```

#!/opt/puppetlabs/puppet/bin/ruby
require 'json'

params = JSON.parse(STDIN.read)

exitcode = 0
params['files'].each do |filename|
  begin
    FileUtils.touch(filename)
    puts "updated file #{filename}"
  rescue
    exitcode = 1
    puts "couldn't update file #{filename}"
  end
end
exit exitcode

```

If your task accepts input on `stdin` it should specify `"input_method": "stdin"` in its `metadata.json` file, or it might not work with `sudo` for some users.

Returning structured output

To return structured data from your task, print only a single JSON object to `stdout` in your task.

Structured output is useful if you want to use the output in another program, or if you want to use the result of the task in a Puppet task plan.

```

#!/usr/bin/env python
import json
import sys
minor = sys.version_info

```

```
result = { "major": sys.version_info.major, "minor":
  sys.version_info.minor }
json.dump(result, sys.stdout)
```

Converting scripts to tasks

To convert an existing script to a task, you can either write a task that wraps the script or you can add logic in your script to check for parameters in environment variables.

If the script is already installed on the target nodes, you can write a task that wraps the script. In the task, read the script arguments as task parameters and call the script, passing the parameters as the arguments.

If the script isn't installed or you want to make it into a cohesive task so that you can manage its version with code management tools, add code to your script to check for the environment variables, prefixed with `PT_`, and read them instead of arguments.



CAUTION: For any tasks that you intend to use with PE and assign RBAC permissions, make sure the script safely handles parameters or validate them to prevent shell injection vulnerabilities.

Given a script that accepts positional arguments on the command line:

```
version=$1
[ -z "$version" ] && echo "Must specify a version to deploy && exit 1

if [ -z "$2" ]; then
  filename=$2
else
  filename=~ /myfile
fi
```

To convert the script into a task, replace this logic with task variables:

```
version=$PT_version #no need to validate if we use metadata
if [ -z "$PT_filename" ]; then
  filename=$PT_filename
else
  filename=~ /myfile
fi
```

Wrapping an existing script

If a script is not already installed on targets and you don't want to edit it, for example if it's a script someone else maintains, you can wrap the script in a small task without modifying it.



CAUTION: For any tasks that you intend to use with PE and assign RBAC permissions, make sure the script safely handles parameters or validate them to prevent shell injection vulnerabilities.

Given a script, `myscript.sh`, that accepts 2 positional args, `filename` and `version`:

1. Copy the script to the module's `files/` directory.
2. Create a metadata file for the task that includes the parameters and file dependency.

```
{ "input_method": "environment", "parameters": { "filename": { "type":
  "String[1]" }, "version": { "type": "String[1]" } }, "files":
  [ "script_example/files/myscript.sh" ] }
```

3. Create a small wrapper task that reads environment variables and calls the task.

```
#!/usr/bin/env bash set -e script_file="$PT_installdir/script_example/
files/myscript.sh" # If this task is going to be run from windows nodes
the wrapper must make sure it's executable chmod +x $script_file
commandline=( "$script_file" "$PT_filename" "$PT_version" ) # If the
```



```
stderr output of the script is important redirect it to stdout.
"${commandline[@]}" 2>&1
```

Supporting no-op in tasks

Tasks support no-operation functionality, also known as no-op mode. This function shows what changes the task would make, without actually making those changes.

No-op support allows a user to pass the `--noop` flag with a command to test whether the task will succeed on all targets before making changes.

To support no-op, your task must include code that looks for the `_noop` metaparameter. No-op is supported only in Puppet Enterprise.

If the user passes the `--noop` flag with their command, this parameter is set to `true`, and your task must not make changes. You must also set `supports_noop` to `true` in your task metadata or the task runner will refuse to run the task in noop mode.

No-op metadata example

```
{
  "description": "Write content to a file.",
  "supports_noop": true,
  "parameters": {
    "filename": {
      "description": "the file to write to",
      "type": "String[1]"
    },
    "content": {
      "description": "The content to write",
      "type": "String"
    }
  }
}
```

No-op task example

```
#!/usr/bin/env python
import json
import os
import sys

params = json.load(sys.stdin)
filename = params['filename']
content = params['content']
noop = params.get('_noop', False)

exitcode = 0

def make_error(msg):
    error = {
        "_error": {
            "kind": "file_error",
            "msg": msg,
            "details": {},
        }
    }
    return error

try:
    if noop:
        path = os.path.abspath(os.path.join(filename, os.pardir))
```

```

file_exists = os.access(filename, os.F_OK)
file_writable = os.access(filename, os.W_OK)
path_writable = os.access(path, os.W_OK)

if path_writable == False:
    exitcode = 1
    result = make_error("Path %s is not writable" % path)
elif file_exists == True and file_writable == False:
    exitcode = 1
    result = make_error("File %s is not writable" % filename)
else:
    result = { "success": True , '_noop': True }
else:
    with open(filename, 'w') as fh:
        fh.write(content)
        result = { "success": True }
except Exception as e:
    exitcode = 1
    result = make_error("Could not open file %s: %s" % (filename, str(e)))
print(json.dumps(result))
exit(exitcode)

```

Task metadata

Task metadata files describe task parameters, validate input, and control how the task runner executes the task.

Your task must have metadata to be published and shared on the Forge. Specify task metadata in a JSON file with the naming convention `<TASKNAME>.json`. Place this file in the module's `./tasks` folder along with your task file.

For example, the module `puppetlabs-mysql` includes the `mysql::sql` task with the metadata file, `sql.json`.

```

{
  "description": "Allows you to execute arbitrary SQL",
  "input_method": "stdin",
  "parameters": {
    "database": {
      "description": "Database to connect to",
      "type": "Optional[String[1]]"
    },
    "user": {
      "description": "The user",
      "type": "Optional[String[1]]"
    },
    "password": {
      "description": "The password",
      "type": "Optional[String[1]]",
      "sensitive": true
    },
    "sql": {
      "description": "The SQL you want to execute",
      "type": "String[1]"
    }
  }
}

```

Adding parameters to metadata

To document and validate task parameters, add the parameters to the task metadata as JSON object, `parameters`.

If a task includes `parameters` in its metadata, the task runner rejects any parameters input to the task that aren't defined in the metadata.

In the `parameter` object, give each parameter a description and specify its Puppet type. For a complete list of types, see the [types documentation](#).

For example, the following code in a metadata file describes a `provider` parameter:

```
"provider": {
  "description": "The provider to use to manage or inspect the service,
  defaults to the system service manager",
  "type": "Optional[String[]]"
}
```

Define default parameters

You can define default task parameters, which are supplied to a task run even if the user does not specify a value for the parameter.

For example, the default location for this `log_location` parameter is `/var/log/puppetlabs`

```
"log_location": {
  "type": "String",
  "description": "The location the log will be stored in"
  "default": "/var/log/puppetlabs"
}
```

Note: Parameters with defaults are considered optional.

Define sensitive parameters

You can define task parameters as sensitive, for example, passwords and API keys. These values are masked when they appear in logs and API responses. When you want to view these values, set the log file to `level: debug`.

To define a parameter as sensitive within the JSON metadata, add the `"sensitive": true` property.

```
{
  "description": "This task has a sensitive property denoted by its
  metadata",
  "input_method": "stdin",
  "parameters": {
    "user": {
      "description": "The user",
      "type": "String[]"
    },
    "password": {
      "description": "The password",
      "type": "String[]",
      "sensitive": true
    }
  }
}
```

Task metadata reference

The following table shows task metadata keys, values, and default values.

Table 1: Task metadata

Metadata key	Description	Value	Default
"description"	A description of what the task does.	String	None

Metadata key	Description	Value	Default
"input_method"	What input method the task runner should use to pass parameters to the task.	<ul style="list-style-type: none"> environment stdin powershell 	Both environment and stdin unless .ps1 tasks, in which case powershell
"parameters"	The parameters or input the task accepts listed with a puppet type string and optional description. See adding parameters to metadata for usage information.	Array of objects describing each parameter	None
"puppet_task_version"	The version of the spec used.	Integer	1 (This is the only valid value.)
"supports_noop"	Whether the task supports no-op mode. Required for the task to accept the <code>--noop</code> option on the command line.	Boolean	False
"implementations"	A list of task implementations and the requirements used to select one to run. See Cross-platform tasks on page 504 for usage information.	Array of Objects describing each implementation	None
"files"	A list of files to be provided when running the task, addressed by module. See Sharing task code on page 505 for usage information.	Array of Strings	None

Task metadata types

Task metadata can accept most Puppet data types.

Restriction:

Some types supported by Puppet can not be represented as JSON, such as `Hash[Integer, String]`, `Object`, or `Resource`. These should not be used in tasks, because they can never be matched.

Table 2: Common task data types

Type	Description
String	Accepts any string.
String[1]	Accepts any non-empty string (a String of at least length 1).
Enum[choice1, choice2]	Accepts one of the listed choices.
Pattern[/\A\w+\Z/]	Accepts Strings matching the regex <code>/\w+/</code> or non-empty strings of word characters.

Type	Description
Integer	Accepts integer values. JSON has no Integer type so this can vary depending on input.
Optional[String[1]]	Optional makes the parameter optional and permits null values. Tasks have no required nullable values.
Array[String]	Matches an array of strings.
Hash	Matches a JSON object.
Variant[Integer, Pattern[/\A\d+\Z/]]	Matches an integer or a String of an integer
Boolean	Accepts Boolean values.

Specifying parameters

Parameters for tasks can be passed to the `bolt` command as CLI arguments or as a JSON hash.

To pass parameters individually to your task or plan, specify the parameter value on the command line in the format `parameter=value`. Pass multiple parameters as a space-separated list. Bolt attempts to parse each parameter value as JSON and compares that to the parameter type specified by the task or plan. If the parsed value matches the type, it is used; otherwise, the original string is used.

For example, to run the `mysql::sql` task to show tables from a database called `mydatabase`:

```
bolt task run mysql::sql database=mydatabase sql="SHOW TABLES" --nodes
neptune --modules ~/modules
```

To pass a string value that is valid JSON to a parameter that would accept both quote the string. For example to pass the string `true` to a parameter of type `Variant[String, Boolean]` use `'foo="true"'`. To pass a String value wrapped in `"` quote and escape it `'string="\val\"'`. Alternatively, you can specify parameters as a single JSON object with the `--params` flag, passing either a JSON object or a path to a parameter file.

To specify parameters as JSON, use the `parameters` flag followed by the JSON: `--params '{ "name": "openssl" }'`

To set parameters in a file, specify parameters in JSON format in a file, such as `params.json`. For example, create a `params.json` file that contains the following JSON:

```
{
  "name": "openssl"
}
```

Then specify the path to that file (starting with an at symbol, `@`) on the command line with the `parameters` flag: `--params @params.json`

Plans in PE

Plans allow you to tie together tasks, scripts, commands, and other plans to create complex workflows with refined access control. You can install modules that contain plans or write your own, then run them from the console or the command line.

Note: If you have set up compilers and you want to use plans, you must either set `master_uris` or you `server_list` on agents to point to your compilers.

A *plan* is a bundle of tasks, commands, scripts, or other plans that can be combined with other logic. They allow you to do complex operations, like running multiple tasks with one command or running certain tasks based on the output of another task.

You can run plans using the tool of your choice: the console, the command line, or the orchestrator API [POST /command/plan_run](#) on page 565 endpoint.

RBAC for plans and tasks do **not** intersect. This means that if a user does not have access to a specific task but they have access to run a plan containing that task, they are still able to run the plan. This allows you to implement more customized access control to tasks by wrapping them within plans. See [Defining plan permissions](#) on page 524 for information about RBAC considerations when writing plans or managing plan access.

Note: If you are running multiple tasks or tasks within plans, make sure your concurrency limit for tasks and bolt-server can accommodate your needs. See [Configure the orchestrator and pe-orchestration-services](#) to adjust these settings.

- [Plans in PE versus Bolt plans](#) on page 518

Some plan language functions, features, and behaviors are different in PE compared to Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

- [Installing plans](#) on page 520

Plans are packaged in modules and must be installed using Code Manager.

- [Running plans in PE](#) on page 521

Use the orchestrator to set up jobs in the console or on the command line and run plans across systems in your infrastructure.

- [Writing plans](#) on page 523

Plans allow you to run more than one task with a single command, compute values for the input to a task, and run other complex workflows at the same time. They also allow you greater flexibility for creating custom RBAC limitations by wrapping tasks and other workflows within plans.

Plans in PE versus Bolt plans

Some plan language functions, features, and behaviors are different in PE compared to Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

Some Bolt modules are not included

The following modules that come pre-installed with open source Bolt are not available by default in PE:

- `ruby_task_helper`
- `python_task_helper`
- `facts`
 - The PE equivalent of `facts` is `factor_task`
- `puppet_agent`
 - The PE equivalent of `puppet_agent` is `pe_bootstrap`

Unavailable plan language functions

The following Bolt plan functions don't work in PE because they haven't been implemented yet or cause issues during plan runs:

- `file::read`
- `file::write`
- `file::readable`
- `file::exists`
- `write_file`
- `add_to_group`
- `remove_from_group`
- `set_config`
- `download_file`

- `get_resources`
- `prompt`
- `resolve_references`
- `run_task_with`
- `resource`
- `set::resources`
- `system::env`

Apply blocks

The apply feature, including `apply_prep`, only works only for targets using the PXP agent and the PCP transport. It fails on remote devices and on targets connected via SSH or WinRM.

Target groups

Support for target groups is unavailable in PE. Using `add_to_group` causes a plan to fail and referencing a group name in `get_targets` doesn't return any nodes. When using `get_targets` you must reference either node certnames or supply a PuppetDB query. Here is an example of a plan using `get_targets` with node certnames:

```
plan example::get_targets_example () {
  $nodes = get_targets(['node1.example.com', 'node2.example.com'])
  run_command('whoami', $nodes)
}
```

Target behaviors

- PE assumes all target references can be matched to either a connected agent with a certname or an entry in the PE inventory service. Target names must match either a certname or an entry in the PE inventory service.
- New targets cannot be added to the inventory service inside a plan. Any new target objects created in a plan won't be able to connect as PE won't recognize them.
- Targets return an empty hash when asked about connection information.

Target configuration

While you can set up node transport configuration through the PE inventory for nodes to use SSH or WinRM, you can't change the configuration settings for targets from within a plan. Using the `set_config` function in a plan causes the plan to fail and referencing a target object's configuration hash always returns an empty hash.

The use of URIs in a target name to override the transport is also not supported. All references to targets (i.e. when using `get_targets`) must be either PuppetDB queries or valid certnames that are already in the PE inventory.

Here is an example of a plan that uses `get_targets` correctly:

```
plan example::get_targets_example () {
  ## NOTE! If you used ssh://node1.example.com as the first entry, this plan
  ## would fail!
  $nodes = get_targets(['node1.example.com', 'node2.example.com'])
  run_command('whoami', $nodes)
}
```

The localhost target

The special target `localhost` is not available for plans in PE. Using `localhost` anywhere in a plan results in a plan failure. If you need to run a plan on the primary server host, use the primary server's certname to reference it.

For example, you can use the following plan for the primary server host `my-primary-server.company.com`:

```
plan example::referencing_the_primary_server(){
```

```
# Note that if you tried to use `localhost` instead of `my-primary-server`
this plan would fail!
run_command('whoami', 'my-primary-server.company.com')
}
```

The `_run_as` parameter

Plans in PE do not support the `_run_as` parameter for changing the user that accesses hosts or executes actions. If this parameter is supplied to any plan function, the plan runs but the user doesn't change.

For example, the following plan is valid, but won't run as `other_user`:

```
plan example::run_as_example (TargetSpec $nodes) {
  run_command('whoami', $nodes, _run_as => 'other_user')
}
```

Script and file sources

When using `run_script` or `download_file`, the source location for the files **must** be from a module that uses a `modulename/filename` selector for a file or directory in `$MODULEROOT/files`. PE does not support file sources that reference absolute paths.

Here is an example of a module structure and a plan that correctly uses the `modulename/filename` selector:

```
example/
### files
  ###my_script.sh
### plans
  ###run_script_example.pp
```

```
plan example::run_script_example (TargetSpec $nodes) {
  run_script('example/my_script.sh', $nodes)
}
```

Code deployment for plans

For plans in PE to work, you must have code manager enabled to deploy code to your primary server.

Primary servers deploy a second `codedir` for plans to load code from. The second code location on primary servers have some effects on standard module functionality:

- Installing modules using the `puppet module install` command doesn't work for plans because the `puppet module` tool won't install to the secondary location for plans. The `puppet module install` command still works for normal Puppet code executed and compiled from Puppet Server.
- A `$modulepath` configuration that uses fully qualified paths might not work for plans if they reference the standard `/etc/puppetlabs/code` location. We recommend using relative paths in `$modulepath`.

Installing plans

Plans are packaged in modules and must be installed using Code Manager.

Unlike tasks, PE doesn't come with any pre-installed plans or modules that contain plans. You must either download modules that contain plans from the Forge or write custom plans.

To install a new module containing a plan, find the module you want on the Forge and follow the **r10k or Code Manager** install instructions under **Start using this module**. The modules containing plans have a **Plans** section in the README.

Some helpful modules with plan content include:

- [puppet_agent](#) - starts a Puppet agent run on targets

- [reboot](#) - manages system reboots

Running plans in PE

Use the orchestrator to set up jobs in the console or on the command line and run plans across systems in your infrastructure.

Running a plan does not update your Puppet configuration. If you run a plan that changes the state of a resource that Puppet is managing (such as upgrading a service or package), a subsequent Puppet run changes the state of that resource back to what is defined in your Puppet configuration.

Note: If you have set up compilers and you want to use plans, you must either set `master_uris` or you `server_list` on agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.

- [Running plans from the console](#) on page 521

Run ad hoc plans from the console.

- [Running plans from the command line](#) on page 521

Run a plan using the `puppet plan run` command.

- [Inspecting plans](#) on page 522

View the plans you have installed and which ones you have permissions to run, as well as individual plan metadata.

Running plans from the console

Run ad hoc plans from the console.

Before you begin

Install the plans you want to use.

Make sure you have the Plans permission to run the plans on the nodes.

When you set up a plan run from the console, the orchestrator creates an ID to track the plan run, shows the nodes included in the plan, and runs the plan on those nodes in the appropriate order. Puppet compiles a new catalog for each node included in the plan run.

1. In the console, in the **Orchestration** section, select **Plans** and then click **Run a plan**.
2. Under **Code environment**, select the environment where you installed the module containing the plan you want to run. For example, **production**.
3. Optional: Under **Job description**, provide a description. This text appears on the **Plans** page.
4. Under **Plan**, select the plan you want to run.
5. Under **Plan parameters**, add optional parameters, then enter values for the optional and required parameters on the list. Click **Add parameter** for each optional parameter-value pair you add to the plan.

To view information about required and optional parameters for the plan, select **view plan metadata** below the **Plan** field.

Express values as strings, arrays, objects, integers, or booleans (true or false). You must express empty strings as two double quotes with no space (" "). Structured values, like an array, must be valid JSON.

Plans with default values run using the default unless you specify another value.

6. Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the plan to run.
7. Select **Run job**.

Your plan status and output appear on the **Plans** page.

Running plans from the command line

Run a plan using the `puppet plan run` command.

On the command line, run the command `puppet plan run` with the following information included:

- The full name of the plan, formatted as `<MODULE> : <PLAN>`.

- Any plan parameters.

Note: To find out what parameters can be included in a plan, view the plan metadata by running the command `puppet plan show <PLAN NAME>` on the command line. For more information, see [Inspecting plans](#) on page 522

- Credentials, if required, formatted with the `--user` and `--password` flags.

For example, if a plan defined in `mymodule/plans/myplan.pp` accepts a `load_balancer` parameter, run:

```
puppet plan run mymodule::myplan load_balancer=lb.myorg.com
```

You can pass a comma-separated list of node names, wildcard patterns, or group names to a plan parameter that is passed to a run function or that the plan resolves using `get_targets`.

Plan command options

The following are common options you can use with the plan action. For a complete list of global options run `puppet plan --help`.

Option	Definition
<code>--params</code>	A string value used to specify either a JSON object that includes the parameters or the path to a JSON file containing the parameters, prefaced with <code>@</code> . For example, <code>@/path/to/file.json</code> . Do not use this flag if specifying inline parameter-value pairs.
<code>--environment</code> or <code>-e</code>	The name of the environment where the plan is installed.
<code>--description</code>	A flag used to provide a description for the job to be shown on the job list and job details pages and returned with the <code>puppet job show</code> command. It defaults to empty.

You can pass parameters into the plan one of two ways:

- Inline, using the `<PARAMETER>=<VALUE>` syntax.
- With the `--params` option, as a JSON object or reference to a JSON file.

For example, review this plan:

```
plan example::test_params(Targetspec $nodes, String $command){
  run_command($command, $nodes)
}
```

You can pass parameters using either option below:

- ```
puppet plan run example::test_params nodes=my-node.company.com
command=whoami
```
- ```
puppet plan run example::test_params --params '{"nodes":"my-
node.company.com", "command":"whoami"}'
```

You can't combine these two ways of passing in parameters. Choose either inline or `--params`.

If you use the inline way, parameter types other than string, integer, double, and Boolean will be interpreted as strings. Use the `--params` method if you want them read as their original type.

Inspecting plans

View the plans you have installed and which ones you have permissions to run, as well as individual plan metadata.

Log into your primary server or client tools workstation and run one of the following commands to see information about your plan inventory:

Command	Definition
<code>puppet plan show</code>	View a list of your permitted plans.
<code>puppet plan show --all</code>	View a list of all installed plans.
<code>puppet plan show <PLAN NAME></code>	View plan metadata. The output includes the plan's required command format and available parameters.

For example, this plan allows a `$nodes` parameter and a `$version` parameter, specified as data types `TargetSpec` and `Integer`.

```
plan infra::upgrade_apache (
  TargetSpec $nodes,
  Integer $version,
){
  run_task('package', $nodes, name => 'apache', action => 'upgrade', version
    => $version)
}
```

To view plan metadata in the console, choose which plan you want to run in the **Plan** field and click on the **View plan metadata** link.

Writing plans

Plans allow you to run more than one task with a single command, compute values for the input to a task, and run other complex workflows at the same time. They also allow you greater flexibility for creating custom RBAC limitations by wrapping tasks and other workflows within plans.

- [Writing plans in Puppet language](#) on page 523

Writing plans in the Puppet language gives you better error handling and more sophisticated control than YAML plans. Plans written in the Puppet language also allow you to apply blocks of Puppet code to remote targets.

- [Writing plans in YAML](#) on page 536

YAML plans run a list of steps in order, which allows you to define simple workflows. Steps can contain embedded Puppet code expressions to add logic where necessary.

Writing plans in Puppet language

Writing plans in the Puppet language gives you better error handling and more sophisticated control than YAML plans. Plans written in the Puppet language also allow you to apply blocks of Puppet code to remote targets.

Naming plans

Name plans according to the module name, file name, and path to ensure code readability.

Place plan files in your module's `./plans` directory, using these file extensions:

- Puppet plans — `.pp`
- YAML plans — `.yaml`, not `.yml`

Plan names are composed of two or more name segments, indicating:

- The name of the module the plan is located in.
- The name of the plan file, without the extension.
- If the plan is in a subdirectory of `./plans`, the path within the module.

For example, given a module called `mymodule` with a plan defined in `./mymodule/plans/myplan.pp`, the plan name is `mymodule::myplan`.

A plan defined in `./mymodule/plans/service/myplan.pp` would be `mymodule::service::myplan`. Use the plan name to refer to the plan when you run commands.

The plan filename `init` is special because the plan it defines is referenced using the module name only. For example, in a module called `mymodule`, the plan defined in `init.pp` is the `mymodule` plan.

Avoid giving plans the same names as constructs in the Puppet language. Although plans don't share their namespace with other language constructs, giving plans these names makes your code difficult to read.

Each plan name segment:

- Must begin with a lowercase letter.
- Can include lowercase letters, digits, or underscores.
- Must not be a [reserved word](#).
- Must not have the same name as any [Puppet data types](#).
- Namespace segments must match the regular expression `\A[a-z][a-z0-9_]*\Z`

Defining plan permissions

RBAC for plans is distinct from RBAC for individual tasks. This distinction means that a user can be excluded from running a certain task, but still have permission to run a plan that contains that task.

The RBAC structure for plans allows you to write plans with more robust, custom control over task permissions. Instead of allowing a user free rein to run a task that can potentially damage your infrastructure, you can wrap a task in a plan and only allow them to run it under circumstances you control.

For example, if you are configuring permissions for a new user to run plan `infra::upgrade_git`, you can allow them to run the `package` task but limit it to the `git` package only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Integer $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version
=> $version)
}
```

Use parameter types to fine-tune access

Parameter types provide another layer of control over user permissions. In the `upgrade_git` example above, the plan only provides access to the `git` package, but the user can choose whatever version of `git` they want. If there are known vulnerabilities in some versions of the `git` package, you can use parameter types like `Enum` to restrict the `version` parameter to versions that are safe enough for deployment.

For example, the `Enum` restricts the `$version` parameter to versions `1:2.17.0-lubuntu1` and `1:2.17.1-lubuntu0.4` only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Enum['1:2.17.0-lubuntu1', '1:2.17.1-lubuntu0.4'] $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version
=> $version)
}
```

You can also use PuppetDB queries to select parameter types.

For example, if you need to restrict the targets that `infra::upgrade_git` can run on, use a PuppetDB query to identify which targets are selected for the `git` upgrade.

```
plan infra::upgrade_git (
  Enum['1:2.17.0-lubuntu1', '1:2.17.1-lubuntu0.4'] $version,
) {
  # Use puppetdb to find the nodes from the "other" team's web cluster
  $query = [from, nodes, ['=', [fact, cluster], "other_team"]]
  $selected_nodes = puppetdb_query($query).map() |$target| {
    $target[certname]
  }
}
```

```

    run_task('package', $selected_nodes, name => 'git', action => 'upgrade',
    version => $version)
  }

```

Specifying plan parameters

Specify plan parameters to do things like determine which targets to run different parts of your plan on. You can pass a parameter as a single target name, comma-separated list of target names, `Target` data type, or array. The target names can be either certnames or inventory node names.

The example plan below shows the target parameters `$load_balancers` and `$webserver`s specified as data type `TargetSpec`. The plan then calls the `run_task` function to specify which targets to run the tasks on. The `Target` names are collected and stored in `$webserver_names` by iterating over the list of `Target` objects returned by `get_targets`. Task parameters are serialized to JSON format so that extracting the names into an array of strings ensures that the `webserver`s parameter is in a format that can be converted to JSON.

```

plan mymodule::my_plan(
  TargetSpec $load_balancer,
  TargetSpec $webserver,
) {

  # Extract the Target name from $webserver
  $webserver_names = get_targets($webserver).map |$n| { $n.name }

  # process webserver
  run_task('mymodule::lb_remove', $load_balancer, webserver =>
  $webserver_names)
  run_task('mymodule::update_frontend_app', $webserver, version => '1.2.3')
  run_task('mymodule::lb_add', $load_balancer, webserver =>
  $webserver_names)
}

```

To execute this plan from the command line, pass the parameters as `parameter=value`. The `Targets`pec accepts either an array as JSON or a comma separated string of target names.

```

puppet plan run mymodule::myplan
load_balancer=lb.myorg.com
webserver='["kermit.myorg.com", "gonzo.myorg.com"]'

```

Alternatively, here is an example of the same plan, run on the same targets, using the [plan run API](#).

```

curl -k -X POST -H "Content-Type: application/json" -H
"X-Authentication:$TOKEN" -d '{ "environment": "$ENV",
"plan_name": "mymodule::myplan", "params": {"targets":
"$TARGET_NAME", "load_balancer": "lb.myorg.com", "webserver":
["kermit.myorg.com", "gonzo.myorg.com"]}}}'
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"

```

Parameters that are passed to the `run_*` plan functions are serialized to JSON. For example, in the plan below, the default value of `$example_nul` is `undef`. The plan calls the `test::demo_undef_bash` with the `example_nul` parameter.

```

plan test::parameter_passing (
  TargetSpec $targets,
  Optional[String[1]] $example_nul = undef,
) {
  return run_task('test::demo_undef_bash', $targets, example_nul =>
  $example_nul)
}

```

The implementation of the `demo_undef_bash.sh` task is:

```
#!/bin/bash
example_env=$PT_example_nul
echo "Environment: $PT_example_nul"
echo "Stdin:"
cat -
```

By default, the task expects parameters passed as a JSON string on `stdin` to be accessible in prefixed environment variables.

Additionally, you can use the [plan_run API](#) with token authentication.

```
curl -k -X POST -H "Content-Type: application/json" -H
"X-Authentication:$TOKEN" -d '{ "environment": "$ENV",
"plan_name": "test::parameter_passing", "params": {"targets":
"$TARGET_NAME"} }'
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"
```

Using Hieradata in plans

Use the `lookup()` function in plans to look up Hieradata. You can look up data inside or outside of apply blocks, or use the `plan_hierarchy` key to look up data both inside and outside apply blocks within the same plan.

Inside [apply blocks](#), PE compiles catalogs for each target and has unlimited access to your Hieradata. You can use the same Hieradata configuration, data, and lookup process as you do throughout PE.

Outside apply blocks, the plan executes a script, doesn't have a concept of a target or context, and cannot load per-target data. These limitations make some common Hieradata features, like interpolating target facts, incompatible with plans in PE outside of apply blocks.

You can look up static Hieradata outside of apply blocks by adding a `plan_hierarchy` key to your Hieradata configuration at the same level as the hierarchy key. This allows you to look up data inside and outside apply blocks in the same plan, enabling you to use your existing Hieradata configuration in plans without encountering an error if per-target interpolations exist and your plan tries to look up data outside an apply block.

Static Hieradata is also useful for user-specific data that you want the plan to look up.

For example, consider the Hieradata configuration below at `<ENV_DIR>/hiera.yaml`.

```
version: 5
hierarchy:
  - name: "Target specific data"
    path: "targets/{trusted.certname}.yaml"
  - name: "Per-OS defaults"
    path: "os/{facts.os.family}.yaml"
  - name: Common
    path: hierarchy.yaml

plan_hierarchy:
  - name: Common
    path: plan_hierarchy.yaml
```

You can set a user-specific API key in the `plan_hierarchy.yaml` data file, as well as use Hieradata to look up a per-target filepath inside an apply block by using the following pieces of data:

Use the following data located at `<ENV_DIR>/data/plan_hierarchy.yaml`:

```
api_key: 12345
```

Use this data located at `<ENV_DIR>/data/targets/myhost.com`:

```
confpath: "C:\Program Files\Common Files\mytool.conf"
```

As a result, the plan looks up the API key in the first `lookup()` call, and the target-specific data inside the `apply` block:

```
plan plan_lookup(
  TargetSpec $targets
) {
  $outside_apply = lookup('api_key')
  run_task("make_request", $targets, 'api_key' => $outside_apply)
  $in_apply = apply($targets) {
    file { ${confpath}:
      ensure => file,
      content => "setting: false"
    }
  }
}
```

Target objects

The `Target` object represents a target and its specific connection options.

The state of a target is stored in the code for the duration of a plan, allowing you to collect facts or set variables for a target and retrieve them later. Target objects must reference a target in the PE inventory. This includes targets connected via the PCP protocol that have puppet-agent installed, or targets in the PE inventory added with either SSH or WinRM credentials or as network devices. Target objects in PE do not have control over their connection information, and the connection info cannot be changed from within a plan.

Because target objects in PE are references, and cannot control their own configuration, accessing target connection info will return empty data.

TargetSpec

The `TargetSpec` type is a wrapper for defining targets that allows you to pass a target, or multiple targets, into a plan. Use `TargetSpec` for plans that accept a set of targets as a parameter to ensure clean interaction with the CLI and other plans.

`TargetSpec` accepts strings allowed by `--targets`, a single target object, or an array of targets and target patterns. To operate on an individual target, resolve the target to a list via `get_targets`.

For example, to loop over each target in a plan, accept a `TargetSpec` argument, but call `get_targets` on it before looping.

```
plan loop(TargetSpec $targets) {
  get_targets($targets).each |$target| {
    run_task('my_task', $target)
  }
}
```

Set variables and facts on targets

You can use the `$target.facts()` and `$target.vars()` functions to set transport configuration values, variables, and facts from a plan. Facts come from running `facter` or another fact collection application on the target, or from a fact store like PuppetDB. Variables are computed externally or assigned directly.

For example, set variables in a plan using `$target.set_var`:

```
plan vars(String $host) {
  $target = get_targets($host)[0]
  $target.set_var('newly_provisioned', true)
  $targetvars = $target.vars
  run_command("echo 'Vars for ${host}: ${$targetvars}'", $host)
}
```

Or set variables in the inventory file using the `vars` key at the group level.

```
groups:
  - name: my_targets
    targets:
      - localhost
    vars:
      operatingsystem: windows
    config:
      transport: ssh
```

Collect facts from targets

The `facts` plan connects to targets, discovers facts, and stores these facts on the targets.

The plan uses these methods to collect facts:

- On `ssh` targets, it runs a Bash script.
- On `winrm` targets, it runs a PowerShell script.
- On `pcp` or targets where the Puppet agent is present, it runs `Facter`.

For example, use the `facts` plan to collect facts and then uses those facts to decide which task to run on the targets.

```
plan run_with_facts(TargetSpec $targets) {
  # This collects facts on targets and update the inventory
  run_plan(facts, targets => $targets)

  $centos_targets = get_targets($targets).filter |$n| { $n.facts['os']
[ 'name' ] == 'CentOS' }
  $ubuntu_targets = get_targets($targets).filter |$n| { $n.facts['os']
[ 'name' ] == 'Ubuntu' }
  run_task(centos_task, $centos_targets)
  run_task(ubuntu_task, $ubuntu_targets)
}
```

Collect facts from PuppetDB

You can use the `puppetdb_fact` plan to collect facts for targets when they are running a Puppet agent and sending facts to PuppetDB.

For example, use the `puppetdb_fact` plan to collect facts, and then use those facts to decide which task to run on the targets.

```
plan run_with_facts(TargetSpec $targets) {
  # This collects facts on targets and update the inventory
  run_plan(puppetdb_fact, targets => $targets)

  $centos_targets = get_targets($targets).filter |$n| { $n.facts['os']
[ 'name' ] == 'CentOS' }
  $ubuntu_targets = get_targets($targets).filter |$n| { $n.facts['os']
[ 'name' ] == 'Ubuntu' }
  run_task(centos_task, $centos_targets)
  run_task(ubuntu_task, $ubuntu_targets)
}
```

Collect general data from PuppetDB

You can use the `puppetdb_query` function in plans to make direct queries to PuppetDB.

For example, you can discover targets from PuppetDB and then run tasks on them. You must configure the PuppetDB client before running it. See the [PQL tutorial](#) to learn how to structure pql queries and see the [PQL reference guide](#) for query examples.

```
plan pdb_discover {
  $result = puppetdb_query("inventory[certname] { app_role ==
'web_server' }")
  # extract the certnames into an array
  $names = $result.map |$r| { $r["certname"] }
  # wrap in url. You can skip this if the default transport is pcp
  $targets = $names.map |$n| { "pcp://${n}" }
  run_task('my_task', $targets)
}
```

Returning results from plans

Use the `return` function to return results that you can use in other plans or save for other uses.

Any plan that does not call the `return` function returns `undef`.

For example,

```
plan return_result(
  $targets
) {
  return run_task('mytask', $targets)
}
```

The result of a plan must match the `PlanResult` type alias. This includes JSON types as well as the plan language types, which have well defined JSON.

- `Undef`
- `String`
- `Numeric`
- `Boolean`
- `Target`
- `ApplyResult`
- `Result`
- `ResultSet`
- `Error`
- Array with only `PlanResult`
- Hash with `String` keys and `PlanResult` values

or

```
Variant[Data, String, Numeric, Boolean, Error, Result, ResultSet, Target,
Array[Boltlib::PlanResult], Hash[String, Boltlib::PlanResult]]
```

Plan errors and failure

Any plan that completes execution without an error is considered successful. There are some specific scenarios that always cause a plan failure, such as calling the `fail_plan` function.

Plan failure due to absent `catch_errors` option

If you call some functions without the `_catch_errors` option and they fail on any target, the plan itself fails. These functions include:

- `upload_file`
- `run_command`
- `run_script`

- `run_task`
- `run_plan`

If there is a plan failure due to an absent `_catch_errors` option when using `run_plan`, any calling plans also halt until a `run_plan` call with `_catch_errors` or a `catch_errors` block is reached.

Failing a plan

If you are writing a plan and think it's failing, you can fail the plan with the `fail_plan` function. This function fails the plan and prevents calling plans from executing any further, unless `run_plan` was called with `_catch_errors` or in a `catch_errors` block.

For example, use the `fail_plan` function to pass an existing error or create a new error with a message that includes the kind, details, or issue code.

```
fail_plan('The plan is failing', 'mymodules/pear-shaped', {'failednodes' =>
  $result.error_set.names})
# or
fail_plan($errorobject)
```

Catching errors in plans

When you use the `catch_errors` function, it executes a block of code and returns any errors, or returns the result of the block if no errors are raised.

Here is an example of the `catch_errors` function.

```
plan test (String[1] $role) {
  $result_or_error = catch_errors(['pe/puppetdb-error']) || {
    puppetdb_query('inventory[certname] { app_role == ${role} }')
  }
  $targets = if $result_or_error =~ Error {
    # If the PuppetDB query fails
    warning("Could not fetch from puppet. Using defaults instead")
    # TargetSpec string
    "all"
  } else {
    $result_or_error
  }
}
```

If there is an error in a plan, it returns the `Error` data type, which includes:

- `msg`: The error message string.
- `kind`: A string that defines the kind of error similar to an error class.
- `details`: A hash with details about the error from a task or from information about the state of a plan when it fails, for example, `exit_code` or `stack_trace`.
- `issue_code`: A unique code for the message that can be used for translation.

Use the `Error` data type in a case expression to match against different kinds of errors. To recover from certain errors and fail on others, set up your plan to include conditionals based on errors that occur while your plan runs. For example, you can set up a plan to retry a task when a timeout error occurs, but fail when there is an authentication error.

The first plan below continues whether it succeeds or fails with a `mymodule/not-serious` error. Other errors cause the plan to fail.

```
plan mymodule::handle_errors {
  $result = run_plan('mymodule::myplan', '_catch_errors' => true)
  case $result {
    Error['mymodule/not-serious'] : {
```

```

    notice("${result.message}")
  }
  Error : { fail_plan($result) } }
  run_plan('mymodule::plan2')
}

```

Puppet and Ruby functions in plans

You can package some common general logic in plans using Puppet language and Ruby functions; however, some functions are not allowed. You can also call plan functions, such as `run_task` or `run_plan`, from within a function.

These Puppet language constructs are not allowed in plans:

- Defined types.
- Classes.
- Resource expressions, such as `file { title: mode => '0777' }`
- Resource default expressions, such as `File { mode => '0666' }`
- Resource overrides, such as `File['/tmp/foo'] { mode => '0444' }`
- Relationship operators: `->` `<-` `~>` `<~`
- Functions that operate on a catalog: `include`, `require`, `contain`, `create_resources`.
- Collector expressions, such as `SomeType <| |>`, `SomeType <<| |>>`
- ERB templates.

Additionally, there are some nuances of the Puppet language to keep in mind when writing plans:

- The `--strict_variables` option is on, so if you reference a variable that is not set, you get an error.
- The `--strict=error` option is on, so minor language issues generate errors. For example `{ a => 10, a => 20 }` is an error because there is a duplicate key in the hash.
- Most Puppet settings are empty and not configurable when using plans in PE.
- Logs include "source location" (file, line) instead of resource type or name.

Handling plan function results

Each *execution function*, or a function you use to operate on one or more targets, returns a `ResultSet`. Each target you executed on returns a `Result`. The `apply` action returns a `ResultSet` containing `ApplyResult` objects.

You can iterate on an instance of `ResultSet` as if it were an `Array[Variant[Result, ApplyResult]]`. This means iterative functions like `each`, `map`, `reduce`, or `filter` work directly on the `ResultSet` returning each result.

A `ResultSet` may contain these functions:

Function	Definition
<code>names()</code>	Names all targets in the set as an <code>Array</code> .
<code>empty()</code>	Returns <code>Boolean</code> if the execution result set is empty.
<code>count()</code>	Returns an <code>Integer</code> count of targets.
<code>first()</code>	Specifies the first <code>Result</code> object, useful to unwrap single results.
<code>find(String \$target_name)</code>	Specifies the <code>Result</code> for a specific target.
<code>error_set()</code>	Returns a <code>ResultSet</code> containing only the results of failed targets.
<code>ok_set()</code>	Returns a <code>ResultSet</code> containing only the successful results.

Function	Definition
<code>filter_set(block)</code>	Filters a <code>ResultSet</code> with the given block and returns a <code>ResultSet</code> object (where the filter function returns an array or hash).
<code>targets()</code>	Specifies an array of all the <code>Target</code> objects from every <code>Result</code> in the set.
<code>ok()</code>	Specifies a Boolean that is the same as <code>error_set.empty</code> .
<code>to_data()</code>	Returns an array of hashes representing either <code>Result</code> or <code>ApplyResults</code> .

A `Result` may contain these functions:

Function	Definition
<code>value()</code>	Specifies the hash containing the value of the <code>Result</code> .
<code>target()</code>	Specifies the <code>Target</code> object that the <code>Result</code> is from.
<code>error()</code>	Returns an <code>Error</code> object constructed from the <code>_error</code> in the value.
<code>message()</code>	Specifies the <code>_output</code> key from the value.
<code>ok()</code>	Returns <code>true</code> if the <code>Result</code> was successful.
<code>[]</code>	Accesses the value hash directly.
<code>to_data()</code>	Returns a hash representation of <code>Result</code> .
<code>action()</code>	Returns a string representation of result type (task, command, etc.).

An `ApplyResult` may contain these functions.

Function	Definition
<code>report()</code>	Returns the hash containing the Puppet report from the application.
<code>target()</code>	Returns the <code>Target</code> object that the <code>Result</code> is from.
<code>error()</code>	Returns an <code>Error</code> object constructed from the <code>_error</code> in the value.
<code>ok()</code>	Returns <code>true</code> if the <code>Result</code> was successful.
<code>to_data()</code>	Returns a hash representation of <code>ApplyResult</code> .
<code>action()</code>	Returns a string representation of result type (apply).

For example, to check if a task ran correctly on all targets, and the check fails if the task fails:

```
$r = run_task('sometask', ..., '_catch_errors' => true)
unless $r.ok {
  fail("Running sometask failed on the targets ${r.error_set.names}")
}
```

You can do iteration and check if the result is an error. This example outputs feedback about the result of a task.

```
$r = run_task('sometask', ..., '_catch_errors' => true)
$r.each |$result| {
  $target = $result.target.name
  if $result.ok {
    out::message("${target} returned a value: ${result.value}")
  } else {
    out::message("${target} errored with a message:
${result.error.message}")
  }
}
```

Similarly, you can iterate over the array of hashes returned by calling `to_data` on a `ResultSet` and access hash values. For example,

```
$r = run_command('whoami')
$r.to_data.each |$result_hash| { notice($result_hash['result']['stdout']) }
```

You can also use `filter_set` to filter a `ResultSet` and apply a `ResultSet` function such as `targets` to the output:

```
$filtered = $result.filter_set |$r| {
  $r['tag'] == "you're it"
}.targets
```

Applying manifest blocks from a plan

You can apply manifest blocks, or chunks of Puppet code, to remote systems during plan execution using the `apply` and `apply_prep` functions.

You can create manifest blocks that use existing content from the Forge, or use a plan to mix procedural orchestration and action with declarative resource configuration from a block. Most features of the Puppet language are available in a manifest block.

If your plan includes a manifest block, use the `apply_prep` function in your plan *before* your manifest block. The `apply_prep` function syncs and caches plugins and gathers facts by running [Facter](#), making the facts available to the manifest block.

For example:

```
apply_prep($target)
apply($target) { notify { foo: } }
```

Note: You can use `apply` and `apply_prep` only on targets connected via PCP.

apply options

The `apply` function supports these options:

Option	Default value	Description
<code>_catch_errors</code>	<code>true</code>	Returns a <code>ResultSet</code> , including failed results, rather than failing the plan. Boolean.
<code>_description</code>	<code>none</code>	Adds a description to the <code>apply</code> block. String.

Option	Default value	Description
<code>_noop</code>	<code>true</code>	Applies the manifest block in no-operation mode, returning a report of changes it would make but does not take action. Boolean.

For example,

```
# Preview installing docker as root on $targets.
apply($targets, _catch_errors => true, _noop => true) {
  include 'docker'
}
```

How manifest blocks are applied

When you apply a manifest code from a plan, the manifest code and any facts generated for each target are sent to Puppet Server for compilation. During code compilation, variables are generated in the following order:

1. Facts gathered from the targets set in your inventory.
2. Local variables from the plan.
3. Variables set in your inventory.

After a successful compilation, PE copies custom module content from the module path and applies the catalog to each target. After the catalog is executed on each target, `apply` generates and returns a report about each target.

Return value

The `apply` function returns a [ResultSet](#) object that contains an [ApplyResult](#) object for each target.

For example:

```
$results = apply($targets) { ... }
$results.each |$result| {
  out::message($result.report)
}
```

Using Hiera data in a manifest block

Hiera is a key-value configuration data look up system, used for separating data from Puppet code. Use Hiera data to implicitly override default class parameters. You can also explicitly look up data from Hiera via the `lookup` function.

Note: Plans in PE currently only support Hiera version 5.

For example:

```
plan do_thing() {
  apply('node1.example.com') {
    notice("Some data in Hiera: ${lookup('mydata')}")
  }
}
```

Plan logging

You can view plan run information in log files or printed to a terminal session using the `out : :message` function or built-in Puppet logging functions.

Outputting to the CLI or console

Use `out : :message` to display output from plans. This function always prints message strings to STDOUT regardless of the log level and doesn't log them to the log file. When using `out : :message` in a plan, the messages are visible on the **Plan details** page in the console.

Puppet log functions

In addition to `out : :message`, you can use Puppet logging functions. Puppet logs messages to `/var/log/puppetlabs/orchestration-services/orchestration-services.log`

When using Puppet logging, each command's usual logging level is downgraded by one level except for `warn` and `error`.

For example, here are the Puppet logging commands with their actual level when used in plans.

```

...
warning('logging text') - logs at warn level
err('logging text') - logs at error level

notice('logging text') - logs at info level
info('logging text') - logs at debug level
debug('logging text') - logs at trace level
...

```

The log level for `orchestration-services.log` is configured with normal levels. for more information about log levels for Bolt, see [Puppet log functions in Bolt](#).

Default action logging

PE logs plan actions through the `upload_file`, `run_command`, `run_script`, or `run_task` functions. By default, it logs an info level message when an action starts and another when it completes. You can pass a description to the function to replace the generic log message.

```
run_task(my_task, $targets, "Better description", param1 => "val")
```

If your plan contains many small actions, you might want to suppress these messages and use explicit calls to the Puppet log functions instead. To do this, wrap actions in a `without_default_logging` block, which logs action messages at info level instead of notice.

For example, you can loop over a series of targets without logging each action.

```

plan deploy( TargetSpec $targets) {
  without_default_logging() || {
    get_targets($targets).each |$target| {
      run_task(deploy, $target)
    }
  }
}

```

To avoid complications with parser ambiguity, always call `without_default_loggingwith ()` and empty block args `||`.

Correct example

```
without_default_logging() || { run_command('echo hi', $targets) }
```

Incorrect example

```
without_default_logging { run_command('echo hi', $targets) }
```

Example plans

Check out some example plans for inspiration when writing your own.

Resource	Description	Level
facts module	Contains tasks and plans to discover facts about target systems.	Getting started
facts plan	Gathers facts using the facts task and sets the facts in inventory.	Getting started
facts::info plan	Uses the facts task to discover facts and map relevant fact values to targets.	Getting started
reboot module	Contains tasks and plans for managing system reboots.	Intermediate
reboot plan	Restarts a target system and waits for it to become available again.	Intermediate
Introducing Masterless Puppet with Bolt	Blog post explaining how plans can be used to deploy a load-balanced web server.	Advanced
profiles::nginx_install plan	Shows an example plan for deploying Nginx.	Advanced

- **Getting started** resources show simple use cases such as running a task and manipulating the results.
- **Intermediate** resources show more advanced features in the plan language.
- **Advanced** resources show more complex use cases such as applying puppet code blocks and using external modules.

Writing plans in YAML

YAML plans run a list of steps in order, which allows you to define simple workflows. Steps can contain embedded Puppet code expressions to add logic where necessary.

Note: YAML plans are an experimental feature and might experience breaking changes in future minor releases.

Naming plans

Name plans according to the module name, file name, and path to ensure code readability.

Place plan files in your module's `./plans` directory, using these file extensions:

- Puppet plans — `.pp`
- YAML plans — `.yaml`, not `.yml`

Plan names are composed of two or more name segments, indicating:

- The name of the module the plan is located in.
- The name of the plan file, without the extension.
- If the plan is in a subdirectory of `./plans`, the path within the module.

For example, given a module called `mymodule` with a plan defined in `./mymodule/plans/myplan.pp`, the plan name is `mymodule::myplan`.

A plan defined in `./mymodule/plans/service/myplan.pp` would be `mymodule::service::myplan`. Use the plan name to refer to the plan when you run commands.

The plan filename `init` is special because the plan it defines is referenced using the module name only. For example, in a module called `mymodule`, the plan defined in `init.pp` is the `mymodule` plan.

Avoid giving plans the same names as constructs in the Puppet language. Although plans don't share their namespace with other language constructs, giving plans these names makes your code difficult to read.

Each plan name segment:

- Must begin with a lowercase letter.
- Can include lowercase letters, digits, or underscores.
- Must not be a [reserved word](#).
- Must not have the same name as any [Puppet data types](#).
- Namespace segments must match the regular expression `\A[a-z][a-z0-9_]*\Z`

Defining plan permissions

RBAC for plans is distinct from RBAC for individual tasks. This distinction means that a user can be excluded from running a certain task, but still have permission to run a plan that contains that task.

The RBAC structure for plans allows you to write plans with more robust, custom control over task permissions. Instead of allowing a user free rein to run a task that can potentially damage your infrastructure, you can wrap a task in a plan and only allow them to run it under circumstances you control.

For example, if you are configuring permissions for a new user to run plan `infra::upgrade_git`, you can allow them to run the `package` task but limit it to the `git` package only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Integer $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version
=> $version)
}
```

Use parameter types to fine-tune access

Parameter types provide another layer of control over user permissions. In the `upgrade_git` example above, the plan only provides access to the `git` package, but the user can choose whatever version of `git` they want. If there are known vulnerabilities in some versions of the `git` package, you can use parameter types like `Enum` to restrict the `version` parameter to versions that are safe enough for deployment.

For example, the `Enum` restricts the `$version` parameter to versions `1:2.17.0-lubuntu1` and `1:2.17.1-lubuntu0.4` only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Enum['1:2.17.0-lubuntu1', '1:2.17.1-lubuntu0.4'] $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version
=> $version)
}
```

You can also use PuppetDB queries to select parameter types.

For example, if you need to restrict the targets that `infra::upgrade_git` can run on, use a PuppetDB query to identify which targets are selected for the `git` upgrade.

```
plan infra::upgrade_git (
  Enum['1:2.17.0-lubuntu1', '1:2.17.1-lubuntu0.4'] $version,
) {
  # Use puppetdb to find the nodes from the "other" team's web cluster
  $query = [from, nodes, ['='], [fact, cluster], "other_team"]]
  $selected_nodes = puppetdb_query($query).map() |$target| {
```

```

    $target[certname]
  }
  run_task('package', $selected_nodes, name => 'git', action => 'upgrade',
    version => $version)
}

```

Plan structure

YAML plans contain a list of steps with optional parameters and results.

YAML maps accept these keys:

- **steps:** The list of steps to perform
- **parameters:** (Optional) The parameters accepted by the plan
- **return:** (Optional) The value to return from the plan

Steps key

The **steps** key is an array of step objects, each of which corresponds to a specific action to take.

When the plan runs, each step is executed in order. If a step fails, the plan halts execution and raises an error containing the result of the step that failed.

Steps use these fields:

- **name:** A unique name that can be used to refer to the result of the step later
- **description:** (Optional) An explanation of what the step is doing

Other available keys depend on the type of step.

Command step

Use a command step to run a single command on a list of targets and save the results, containing stdout, stderr, and exit code.

The step fails if the exit code of any command is non-zero.

Command steps use these fields:

- **command:** The command to run
- **target:** A target or list of targets to run the command on

For example:

```

steps:
  - command: hostname -f
    target:
      - web1.example.com
      - web2.example.com
      - web3.example.com
    description: "Get the webserver hostnames"

```

Task step

Use a task step to run a Bolt task on a list of targets and save the results.

Task steps use these fields:

- **task:** The task to run
- **target:** A target or list of targets to run the task on
- **parameters:** (Optional) A map of parameter values to pass to the task

For example:

```

steps:
  - task: package

```

```

target:
  - web1.example.com
  - web2.example.com
  - web3.example.com
description: "Check the version of the openssl package on the
webservers"
parameters:
  action: status
  name: openssl

```

Script step

Use a script step to run a script on a list of targets and save the results.

The script must be in the `files/` directory of a module. The name of the script must be specified as `<modulename>/path/to/script`, omitting the `files` directory from the path.

Script steps use these fields:

- `script`: The script to run
- `target`: A target or list of targets to run the script on
- `arguments`: (Optional) An array of command-line arguments to pass to the script

For example:

```

steps:
  - script: mymodule/check_server.sh
    target:
      - web1.example.com
      - web2.example.com
      - web3.example.com
    description: "Run mymodule/files/check_server.sh on the webservers"
    arguments:
      - "/index.html"
      - 60

```

File upload step

Use a file upload step to upload a file to a specific location on a list of targets.

The file to upload must be in the `files/` directory of a Puppet module. The source for the file must be specified as `<modulename>/path/to/file`, omitting the `files` directory from the path.

File upload steps use these fields:

- `source`: The location of the file to be uploaded
- `destination`: The location to upload the file to

For example:

```

steps:
  - source: mymodule/motd.txt
    destination: /etc/motd
    target:
      - web1.example.com
      - web2.example.com
      - web3.example.com
    description: "Upload motd to the webservers"

```

Plan step

Use a plan step to run another plan and save its result.

Plan steps use these fields:

- `plan`: The name of the plan to run
- `parameters`: (Optional) A map of parameter values to pass to the plan

For example:

```
steps:
  - plan: facts
    description: "Gather facts for the webserver using the built-in facts plan"
    parameters:
      nodes:
        - web1.example.com
        - web2.example.com
        - web3.example.com
```

Resources step

Use a `resources` step to apply a list of Puppet resources. A resource defines the desired state for part of a target. Bolt ensures each resource is in its desired state. Like the steps in a plan, if any resource in the list fails, the rest are skipped.

For each `resources` step, Bolt executes the `apply_prep` plan function against the targets specified with the `targets` field. For more information about `apply_prep` see the Applying manifest block section.

Resources steps use these fields:

- `resources`: An array of resources to apply
- `target`: A target or list of targets to apply the resources on

Each resource is a YAML map with a `type` and `title`, and optionally a `parameters` key. The resource type and title can either be specified separately with the `type` and `title` keys, or can be specified in a single line by using the type name as a key with the title as its value.

For example:

```
steps:
  - resources:
    # This resource is type 'package' and title 'nginx'
    - package: nginx
      parameters:
        ensure: latest
    # This resource is type 'service' and title 'nginx'
    - type: service
      title: nginx
      parameters:
        ensure: running
    target:
      - web1.example.com
      - web2.example.com
      - web3.example.com
    description: "Set up nginx on the webserver"
```

Parameters key

Plans accept parameters with the `parameters` key. The value of `parameters` is a map, where each key is the name of a parameter and the value is a map describing the parameter.

Parameter values can be referenced from steps as variables.

Parameters use these fields:

- `type`: (Optional) A valid [Puppet data type](#). The value supplied must match the type or the plan fails.

- `default:` (Optional) Used if no value is given for the parameter
- `description:` (Optional)

For example, this plan accepts a `load_balancer` name as a string, two sets of nodes called `frontends` and `backends`, and a `version` string:

```
parameters:
  # A simple parameter definition doesn't need a type or description
  load_balancer:
  frontends:
    type: TargetSpec
    description: "The frontend web servers"
  backends:
    type: TargetSpec
    description: "The backend application servers"
  version:
    type: String
    description: "The new application version to deploy"
```

How strings are evaluated

The behavior of strings is defined by how they're written in the plan.

'single-quoted strings' are treated as string literals without any interpolation.

"double-quoted strings" are treated as Puppet language double-quoted strings with variable interpolation.

| block-style strings are treated as expressions of arbitrary Puppet code. Note the string itself must be on a new line after the | character.

bare strings are treated dynamically based on their content. If they begin with a \$, they're treated as Puppet code expressions. Otherwise, they're treated as YAML literals.

Here's an example of different kinds of strings in use:

```
parameters:
  message:
    type: String
    default: "hello"

steps:
  - eval: hello
    description: 'This will evaluate to: hello'
  - eval: $message
    description: 'This will evaluate to: hello'
  - eval: '$message'
    description: 'This will evaluate to: $message'
  - eval: "${message} world"
    description: 'This will evaluate to: hello world'
  - eval: |
    [$message, $message, $message].join(" ")
    description: 'This will evaluate to: hello hello hello'
```

Using variables and simple expressions

The simplest way to use a variable is to reference it directly by name. For example, this plan takes a parameter called `nodes` and passes it as the target list to a step:

```
parameters:
  nodes:
    type: TargetSpec

steps:
  - command: hostname -f
```

```
target: $nodes
```

Variables can also be interpolated into string values. The string must be double-quoted to allow interpolation. For example:

```
parameters:
  username:
    type: String

steps:
  - task: echo
    message: "hello ${username}"
    target: $nodes
```

Many operations can be performed on variables to compute new values for step parameters or other fields.

Indexing arrays or hashes

You can retrieve a value from an Array or a Hash using the `[]` operator. This operator can also be used when interpolating a value inside a string.

```
parameters:
  users:
    # Array[String] is a Puppet data type representing an array of strings
    type: Array[String]

steps:
  - task: user::add
    target: 'host.example.com'
    parameters:
      name: $users[0]
  - task: echo
    target: 'host.example.com'
    parameters:
      message: "hello ${users[0]}"
```

Calling functions

You can call a built-in [Bolt function](#) or [Puppet function](#) to compute a value.

```
parameters:
  users:
    type: Array[String]

steps:
  - task: user::add
    parameters:
      name: $users.first
  - task: echo
    message: "hello ${users.join(',')}"
```

Using code blocks

Some Puppet functions take a block of code as an argument. For instance, you can filter an array of items based on the result of a block of code.

The result of the `filter` function is an array here, not a string, because the expression isn't inside quotes

```
parameters:
  numbers:
    type: Array[Integer]
```

```

steps:
  - task: sum
    description: "add up the numbers > 5"
    parameters:
      indexes: $numbers.filter |$num| { $num > 5 }

```

Connecting steps

You can connect multiple steps by using the result of one step to compute the parameters for another step.

name key

The name key makes its results available to later steps in a variable with that name.

This example uses the map function to get the value of stdout from each command result and then joins them into a single string separated by commas.

```

parameters:
  nodes:
    type: TargetSpec

steps:
  - name: hostnames
    command: hostname -f
    target: $nodes
  - task: echo
    parameters:
      message: $hostnames.map |$hostname_result|
      { $hostname_result['stdout'] }.join(',')

```

eval step

The eval step evaluates an expression and saves the result in a variable. This is useful to compute a variable to use multiple times later.

```

parameters:
  count:
    type: Integer

steps:
  - name: double_count
    eval: $count * 2
  - task: echo
    target: webl.example.com
    parameters:
      message: "The count is ${count}, and twice the count is
      ${double_count}"

```

Returning results

You can return a result from a plan by setting the return key at the top level of the plan. When the plan finishes, the return key is evaluated and returned as the result of the plan. If no return key is set, the plan returns undef

```

steps:
  - name: hostnames
    command: hostname -f
    target: $nodes

return: $hostnames.map |$hostname_result| { $hostname_result['stdout'] }

```

Applying manifest blocks from a plan

You can apply manifest blocks, or chunks of Puppet code, to remote systems during plan execution using the `apply` and `apply_prep` functions.

You can create manifest blocks that use existing content from the Forge, or use a plan to mix procedural orchestration and action with declarative resource configuration from a block. Most features of the Puppet language are available in a manifest block.

If your plan includes a manifest block, use the `apply_prep` function in your plan *before* your manifest block. The `apply_prep` function syncs and caches plugins and gathers facts by running [Facter](#), making the facts available to the manifest block.

For example:

```
apply_prep($target)
apply($target) { notify { foo: } }
```

Note: You can use `apply` and `apply_prep` only on targets connected via PCP.

apply options

The `apply` function supports these options:

Option	Default value	Description
<code>_catch_errors</code>	<code>true</code>	Returns a <code>ResultSet</code> , including failed results, rather than failing the plan. Boolean.
<code>_description</code>	<code>none</code>	Adds a description to the <code>apply</code> block. String.
<code>_noop</code>	<code>true</code>	Applies the manifest block in no-operation mode, returning a report of changes it would make but does not take action. Boolean.

For example,

```
# Preview installing docker as root on $targets.
apply($targets, _catch_errors => true, _noop => true) {
  include 'docker'
}
```

How manifest blocks are applied

When you apply a manifest code from a plan, the manifest code and any facts generated for each target are sent to Puppet Server for compilation. During code compilation, variables are generated in the following order:

1. Facts gathered from the targets set in your inventory.
2. Local variables from the plan.
3. Variables set in your inventory.

After a successful compilation, PE copies custom module content from the module path and applies the catalog to each target. After the catalog is executed on each target, `apply` generates and returns a report about each target.

Return value

The `apply` function returns a [ResultSet](#) object that contains an [ApplyResult](#) object for each target.

For example:

```
$results = apply($targets) { ... }
$results.each |$result| {
  out::message($result.report)
}
```

Using Hieradata in a manifest block

Hiera is a key-value configuration data look up system, used for separating data from Puppet code. Use Hiera data to implicitly override default class parameters. You can also explicitly look up data from Hiera via the lookup function.

Note: Plans in PE currently only support Hiera version 5.

For example:

```
plan do_thing() {
  apply('node1.example.com') {
    notice("Some data in Hiera: ${lookup('mydata')}")
  }
}
```

Computing complex values

To compute complex values, you can use a Puppet code expression as the value of any field of a step except the name.

Bolt loads the plan as a YAML data structure. As it executes each step, it evaluates any expressions embedded in the step. Each plan parameter and the values of every previous named step are available in scope.

This lets you take advantage of the power of Puppet language in the places it's necessary, while keeping the rest of your plan simple.

When your plans need more sophisticated control flow or error handling beyond running a list of steps in order, it's time to convert them to [Puppet Language plans](#).

Converting YAML plans to Puppet language plans

You can convert a YAML plan to a Puppet language plan with the `bolt plan convert` command.

```
bolt plan convert path/to/my/plan.yaml
```

This command takes the relative or absolute path to the YAML plan to be converted and prints the converted Puppet language plan to stdout.

Note: Converting a YAML plan might result in a Puppet plan which is syntactically correct, but behaves differently. Always manually verify a converted Puppet language plan's functionality. There are some constructs that do not translate from YAML plans to Puppet language plans. These are listed [TODO: insert link to section below!] below. If you convert a YAML plan to Puppet and it changes behavior, [file an issue](#) in Bolt's Git repo.

For example, with this YAML plan:

```
# site-modules/mymodule/plans/yamlplan.yaml
parameters:
  nodes:
    type: TargetSpec
steps:
  - name: run_task
    task: sample
    target: $nodes
    parameters:
      message: "hello world"
return: $run_task
```

Run the following conversion:

```
$ bolt plan convert site-modules/mymodule/plans/yamlplan.yaml
# WARNING: This is an autogenerated plan. It may not behave as expected.
plan mymodule::yamlplan(
  TargetSpec $nodes
) {
  $run_task = run_task('sample', $nodes, {'message' => "hello world"})
  return $run_task
}
```

Quirks when converting YAML plans to Puppet language

There are some quirks and limitations associated with converting a plan expressed in YAML to a plan expressed in the Puppet language. In some cases it is impossible to accurately translate from YAML to Puppet. In others, code that is generated from the conversion is syntactically correct but not idiomatic Puppet code.

Named `eval` step

The `eval` step allows snippets of Puppet code to be expressed in YAML plans. When converting a multi-line `eval` step to Puppet code and storing the result in a variable, use the `with lambda`.

For example, here is a YAML plan with a multi-line `eval` step:

```
parameters:
  foo:
    type: Optional[Integer]
    description: foo
    default: 0

steps:
  - eval: |
      $x = $foo + 1
      $x * 2
    name: eval_step

return: $eval_step
```

And here is the same plan, converted to the Puppet language:

```
plan yaml_plans::with_lambda(
  Optional[Integer] $foo = 0
) {
  $eval_step = with() || {
    $x = $foo + 1
    $x * 2
  }

  return $eval_step
}
```

Writing this plan from scratch using the Puppet language, you would probably not use the `lambda`. In this example the converted Puppet code is correct, but not as natural or readable as it could be.

Resource step variable interpolation

When applying Puppet resources in a resource step, variable interpolation behaves differently in YAML plans and Puppet language plans. To illustrate this difference, consider this YAML plan:

```
steps:
  - target: localhost
    description: Apply a file resource
```

```

resources:
- type: file
  title: '/tmp/foo'
  parameters:
    content: $facts['os']['family']
    ensure: present
- name: file_contents
  description: Read contents of file managed with file resource
  eval: >
    file::read('/tmp/foo')

return: $file_contents

```

This plan performs `apply_prep` on a `localhost` target. Then it uses a Puppet `file` resource to write the OS family discovered from the Puppet `$facts` hash to a temporary file. Finally, it reads the value written to the file and returns it. Running `bolt plan convert` on this plan produces this Puppet code:

```

plan yaml_plans::interpolation_pp() {
  apply_prep('localhost')
  $interpolation = apply('localhost') {
    file { '/tmp/foo':
      content => $facts['os']['family'],
      ensure => 'present',
    }
  }
  $file_contents = file::read('/tmp/foo')

  return $file_contents
}

```

This Puppet language plan works as expected, whereas the YAML plan it was converted from fails. The failure stems from the `$facts` variable being resolved as a plan variable, instead of being evaluated as part of compiling the manifest code in an `applyblock`.

Dependency order

The resources in a `resources` list are applied in order. It is possible to set dependencies explicitly, but when doing so you must refer to them in a particular way. Consider the following YAML plan:

```

parameters:
  nodes:
    type: TargetSpec
steps:
- name: pkg
  target: $nodes
  resources:
    - title: openssh-server
      type: package
      parameters:
        ensure: present
        before: File['/etc/ssh/sshd_config']
    - title: /etc/ssh/sshd_config
      type: file
      parameters:
        ensure: file
        mode: '0600'
        content: ''
        require: Package['openssh-server']

```

Executing this plan fails during catalog compilation because of how Bolt parses the resources referenced in the `before` and `require` parameters. You will see the error message `Could not find resource 'File[/etc/ssh/sshd_config]' in parameter 'before'`. The solution is to not quote the resource titles:

```
parameters:
  nodes:
    type: TargetSpec
steps:
  - name: pkg
    target: $nodes
    resources:
      - title: openssh-server
        type: package
        parameters:
          ensure: present
          before: File[/etc/ssh/sshd_config]
      - title: /etc/ssh/sshd_config
        type: file
        parameters:
          ensure: file
          mode: '0600'
          content: ''
          require: Package[openssh-server]
```

In general, declare resources in order. This is an unusual example to illustrate a case where parameter parsing leads to non-intuitive results.

Puppet orchestrator API v1 endpoints

Use this API to gather details about the orchestrator jobs you run.

- [Puppet orchestrator API: forming requests](#) on page 549

Instructions on interacting with this API.

- [Puppet orchestrator API: command endpoint](#) on page 550

Use the `/command` endpoint to start and stop orchestrator jobs for tasks and plans.

- [Puppet orchestrator API: events endpoint](#) on page 568

Use the `/events` endpoint to learn about events that occurred during an orchestrator job.

- [Puppet orchestrator API: inventory endpoint](#) on page 574

Use the `/inventory` endpoint to discover which nodes can be reached by the orchestrator.

- [Puppet orchestrator API: jobs endpoint](#) on page 576

Use the `/jobs` endpoint to examine orchestrator jobs and their details.

- [Puppet orchestrator API: scheduled jobs endpoint](#) on page 587

Use the `/scheduled_jobs` endpoint to gather information about orchestrator jobs scheduled to run.

- [Puppet orchestrator API: plans endpoint](#) on page 593

Use the `/plans` endpoint to see all known plans in your environments.

- [Puppet orchestrator API: plan jobs endpoint](#) on page 595

Use the `/plan_jobs` endpoint to view details about plan jobs you have run.

- [Puppet orchestrator API: tasks endpoint](#) on page 603

Use the `/tasks` endpoint to view details about the tasks pre-installed by PE and those you've installed.

- [Puppet orchestrator API: root endpoint](#) on page 606

Use the `/orchestrator` endpoint to return metadata about the orchestrator API.

- [Puppet orchestrator API: usage endpoint](#) on page 607

Use the `/usage` endpoint to view details about the active nodes in your deployment.

- [Puppet orchestrator API: scopes endpoint](#) on page 608

Use the `scopes` endpoint to retrieve resources for task constraints.

- [Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Related information

[API index](#) on page 26

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

Puppet orchestrator API: forming requests

Instructions on interacting with this API.

By default, the orchestrator service listens on port 8143, and all endpoints are relative to the `/orchestrator/v1` path. So, for example, the full URL for the `jobs` endpoint on localhost would be `https://localhost:8143/orchestrator/v1/jobs`.

Tip: The orchestrator API accepts well-formed HTTP(S) requests.

Authenticating to the orchestrator API with a token

You need to authenticate requests to the orchestrators's API. You can do this using user authentication tokens.

For detailed information about authentication tokens, see [Token-based authentication](#) on page 218. The following example shows how to use a token in an API request.

To use the `jobs` endpoint of the orchestrator API to get a list of all jobs that exist in the orchestrator, along with their associated metadata, you'd first generate a token with the `puppet-access` tool. You'd then copy that token and replace `<TOKEN>` with that string in the following request:

```
auth_header="X-Authentication: $(puppet-access show) "
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs"

curl --insecure --header "$auth_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Example token usage: deploy an environment

If you want to deploy an environment with the orchestrator's API, you can form a request with the token you generated earlier. For example:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/
deploy"
data='{ "environment": "production", "scope" : { "node_group" :
  "00000000-0000-4000-8000-000000000000" } }'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

This returns a JSON structure that includes a link to the new job started by the orchestrator.

```
{
  "job" : {
    "id" : "https://orchestrator.vm:8143/orchestrator/v1/jobs/81",
    "name" : "81"
  }
}
```

You can make an additional request to get more information about the job. For example:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs/81"

curl --insecure --header "$auth_header" "$uri"
```

Related information

[Token-based authentication](#) on page 218

Authentication tokens allow a user to enter their credentials once, then receive an alphanumeric "token" to use to access different services or parts of the system infrastructure. Authentication tokens are tied to the permissions granted to the user through RBAC, and provide the user with the appropriate access to HTTP requests.

Puppet orchestrator API: command endpoint

Use the `/command` endpoint to start and stop orchestrator jobs for tasks and plans.

POST /command/deploy

Run the orchestrator across all nodes in an environment.

Request format

The request body must be a JSON object using these keys:

Key	Definition
environment	The environment to deploy. This key is required.
scope	Object, required unless <code>target</code> is specified. The PuppetDB query, a list of nodes, a classifier node group id, or an application/application instance to deploy.
description	String, a description of the job.

Key	Definition
<code>noop</code>	Boolean, whether to run the agent in no-op mode. The default is <code>false</code> .
<code>no_noop</code>	Boolean, whether to run the agent in enforcement mode. Defaults to <code>false</code> . This flag overrides <code>noop = true</code> if set in the agent's <code>puppet.conf</code> , and cannot be set to <code>true</code> at the same time as the <code>noop</code> flag.
<code>concurrency</code>	The maximum number of nodes to run at one time. The default is the range between 1 and up to the value set by the following parameter, which defaults to 8: <code>puppet_enterprise::profile::orchestrator::global_</code>
<code>enforce_environment</code>	Boolean, whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be <code>false</code> if <code>environment</code> is an empty string
<code>debug</code>	Boolean, whether to use the <code>--debug</code> flag on Puppet agent runs.
<code>trace</code>	Boolean, whether to use the <code>--trace</code> flag on Puppet agent runs.
<code>evaltrace</code>	Boolean, whether to use the <code>--evaltrace</code> flag on Puppet agent runs.
<code>filetimeout</code>	Integer, sets the <code>--filetimeout</code> flag on Puppet agent runs to the provided value.
<code>http_connect_timeout</code>	Integer, sets the <code>--http_connect_timeout</code> flag on Puppet agent runs to the provided value.
<code>http_keepalive_timeout</code>	Integer, sets the <code>--http_keepalive_timeout</code> flag on Puppet agent runs to the provided value.
<code>http_read_timeout</code>	Integer, sets the <code>--http_read_timeout</code> flag on Puppet agent runs to the provided value.
<code>ordering</code>	String, sets the <code>--ordering</code> flag on Puppet agent runs to the provided value.
<code>skip_tags</code>	String, sets the <code>--skip_tags</code> flag on Puppet agent runs to the provided value.
<code>tags</code>	String, sets the <code>--tags</code> flag on Puppet agent runs to the provided value.
<code>use_cached_catalog</code>	Boolean, whether to use the <code>--use_cached_catalog</code> flag on Puppet agent runs.
<code>usecacheonfailure</code>	Boolean, whether to use the <code>--usecacheonfailure</code> flag on Puppet agent runs.
<code>userdata</code>	An object of arbitrary key/value data supplied to the job.

For example, to deploy the `node1.example.com` environment in no-op mode, the following request is valid:

```
{
  "environment" : "",
  "enforce_environment": false,
  "noop" : true,
  "scope" : {
```

```

    "nodes" : [ "node1.example.com" ]
  },
  "userdata": {
    "servicenow_ticket": "INC0011211"
  }
}

```

Scope

Scope is a JSON object containing exactly one of these keys:

Key	Definition
application	The name of an application or application instance to deploy. If an application type is specified, all instances of that application are deployed.
nodes	A list of node names to target.
query	A PuppetDB or PQL query to use to discover nodes. The target is built from <code>certname</code> values collected at the top level of the query.
node_group	A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group.

To deploy an application instance in the production environment:

```

{
  "environment" : "production",
  "scope" : {
    "application" : "Wordpress_app[demo]"
  }
}

```

To deploy a list of nodes:

```

{
  "environment" : "production",
  "scope" : {
    "nodes" : [ "node1.example.com", "node2.example.com" ]
  }
}

```

To deploy a list of nodes with the `certname` value matching a regex:

```

{
  "environment" : "production",
  "scope" : {
    "query" : [ "from", "nodes", [ "~", "certname", ".*" ] ]
  }
}

```

To deploy to the nodes defined by the "All Nodes" node group:

```

{
  "environment" : "production",
  "scope" : {

```



```

    "node_group" : "00000000-0000-4000-8000-000000000000"
  }
}

```

Response format

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response is a JSON object containing a link to retrieve information about the status of the job and uses any one of these keys:

Key	Definition
id	An absolute URL that links to the newly created job.
name	The name of the newly created job.

For example:

```

{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234"
    "name" : "1234"
  }
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/unknown-environment	If the environment does not exist, the server returns a 404 response.
puppetlabs.orchestrator/empty-environment	If the environment requested contains no applications or no nodes, the server returns a 400 response.
puppetlabs.orchestrator/empty-target	If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.
puppetlabs.orchestrator/dependency-cycle	If the application code contains a cycle, the server returns a 400 response.
puppetlabs.orchestrator/puppetdb-error	If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.
puppetlabs.orchestrator/query-error	If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.

Related information

[Puppet orchestrator API: forming requests](#) on page 549

Instructions on interacting with this API.

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

POST /command/schedule_deploy

Schedule a Puppet run on a set of nodes.

Request format

The request body must be a JSON object. The following keys are required:

Key	Definition
environment	The environment to deploy. Required string.
scope	A PuppetDB query, a list of nodes, or a classifier node group ID to deploy. Required object.
scheduled_time	Timestamp for when to run the scheduled job. If the timestamp is in the past, a 400 error will be thrown. Required ISO-8601 timestamp.

The following keys are optional:

Key	Definition
concurrency	The maximum number of nodes to run at once. The default, if unspecified, is unlimited. Integer.
debug	Whether to use the <code>--debug</code> flag on Puppet agent runs. Boolean.
description	Description of the job. String.
enforce_environment	Whether to force agents to run in the same environment in which their assigned applications are defined. This key is required to be <code>false</code> if <code>environment</code> is an empty string. Boolean.
evaltrace	Whether to use the <code>--evaltrace</code> flag on Puppet agent runs. Boolean.
filetimeout	Sets the <code>--filetimeout</code> flag on Puppet agent runs to the provided value. Integer.
http_connect_timeout	Sets the <code>--http_connect_timeout</code> flag on Puppet agent runs to the provided value. Integer.
http_keepalive_timeout	Sets the <code>--http_keepalive_timeout</code> flag on Puppet agent runs to the provided value. Integer.
http_read_timeout	Sets the <code>--http_read_timeout</code> flag on Puppet agent runs to the provided value. Integer.
noop	Whether to run the agent in no-op mode. Defaults to <code>false</code> . Boolean.
no_noop	Whether to run the agent in enforcement mode. Defaults to <code>false</code> . This flag overrides <code>noop = true</code> if set in the agent's configuration and cannot be set to <code>true</code> at the same time as the <code>noop</code> flag. Boolean.
ordering	Sets the <code>--ordering</code> flag to the provided value on Puppet agent runs. String.

Key	Definition
scheduled_time	The timestamp for when to run the scheduled job. If the timestamp is in the past, a 400 error will be thrown. Required ISO-8601 timestamp.
schedule_options	Object.
interval	Object.
units	Enum in seconds.
value	Positive integer.
skip_tags	Sets the <code>--skip_tags</code> flag to the provided value on Puppet agent runs. String.
tags	Sets the <code>--tags</code> flag to the provided value on Puppet agent runs. String.
trace	Whether to use the <code>--trace</code> flag on Puppet agent runs. Boolean.
use_cached_catalog	Whether to use the <code>--use_cached_catalog</code> flag on Puppet agent runs. Boolean.
usecacheonfailure	Whether to use the <code>--usecacheonfailure</code> flag on Puppet agent runs. Boolean.

To deploy `node1.example.com` in noop mode, the following request is valid:

```
{
  "environment" : "",
  "enforce_environment": false,
  "noop" : true,
  "scope" : {
    "nodes" : ["node1.example.com"]
  },
  "scheduled_time": "2027-05-05T19:50:08Z",
  "schedule_options": {
    "interval": {
      "units": "seconds",
      "value": 86400
    }
  }
}
```

Response format

If all node runs succeed, and the environment is successfully deployed, the server returns a 202 response.

The response will be a JSON object containing a link to retrieve information about the status of the job. The keys of this object are as follows:

Key	Definition
id	An absolute URL that links to the newly created job.
name	The name of the newly created job.

For example:

```
{
```

```

  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1234",
    "name" : "1234"
  }
}

```

Error responses

See the [error response documentation](#) for the general format of error responses. For this endpoint, the `kind` key of the error displays the conflict.

Error	Definition
<code>puppetlabs.orchestrator/unknown-environment</code>	If the environment does not exist, the server returns a 404 response.
<code>puppetlabs.orchestrator/empty-environment</code>	If the environment requested contains no applications or no nodes, the server returns a 400 response.
<code>puppetlabs.orchestrator/empty-target</code>	If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.
<code>puppetlabs.orchestrator/dependency-cycle</code>	If the application code contains a cycle, the server returns a 400 response.
<code>puppetlabs.orchestrator/puppetdb-error</code>	If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.
<code>puppetlabs.orchestrator/query-error</code>	If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.

POST /command/stop

Stop an orchestrator job that is currently in progress. Puppet agent runs that are in progress finish, but no new agent runs start. While agents are finishing, the server continues to produce events for the job.

The job transitions to status `stopped` when all pending agent runs have finished.

This command is *idempotent*: it can be issued against the same job any number of times.

Request format

The JSON body of the request must contain the ID of the job to stop. The job ID is the same value as the `name` property returned with the `deploy` command.

- `job`: the name of the job to stop.

For example:

```

{
  "job": "1234"
}

```

Response format

If the job is stopped successfully, the server returns a 202 response. The response uses these keys:

Key	Definition
<code>id</code>	An absolute URL that links to the newly created job.

Key	Definition
name	The name of the newly created job.
nodes	A hash that shows all of the possible node statuses, and how many nodes are currently in that status.

For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1234",
    "name" : "1234",
    "nodes" : {
      "new" : 5,
      "running" : 8,
      "failed" : 3,
      "errored" : 1,
      "skipped" : 2,
      "finished" : 5
    }
  }
}
```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If a job name is not valid, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If a job name is unknown, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

POST /command/task

Run a permitted task job across a set of nodes. Any nodes specified in the scope that you do not have permission to run tasks on are excluded.

Request format

The request body must be a JSON object and uses the following keys:

Key	Definition
environment	The environment to load the task from. The default is production.
scope	The PuppetDB query, list of nodes, or a node group ID. Application scopes are not allowed for task jobs. This key is required.
description	A description of the job.

Key	Definition
noop	Whether to run the job in no-op mode. The default is <code>false</code> .
task	The task to run on the targets. This key is required.
params	The parameters to pass to the task. This key is required, but can be an empty object.
targets	A collection of target objects used for running the task on nodes through SSH or WinRM via Bolt server.
userdata	An object of arbitrary key/value data supplied to the job.

For example, to run the `package` task on `node1.example.com`, the following request is valid:

```
{
  "environment" : "test-env-1",
  "task" : "package",
  "params" : {
    "action" : "install",
    "name" : "httpd"
  },
  "scope" : {
    "nodes" : [ "node1.example.com" ]
  }
}
```

Scope

Scope is a JSON object containing exactly one of the following keys:

Key	Definition
application	The name of an application or application instance to deploy. If an application type is specified, all instances of that application will be deployed. This key is only supported for the <code>deploy</code> command.
nodes	An array of node names to target.
query	A PuppetDB or PQL query to use to discover nodes. The target is built from the <code>certname</code> values collected at the top level of the query.
node_group	A classifier node group ID. The ID must correspond to a node group that has defined rules. It is not sufficient for parent groups of the node group in question to define rules. The user must also have permissions to view the node group. Any nodes specified in the scope that the user does not have permissions to run the task on are excluded.

To deploy an application instance in the production environment, the following request is valid:

```
{
  "environment" : "production",
  "scope" : {
    "application" : "Wordpress_app[demo]"
  }
}
```

```
}
```

To run a task on a list of nodes, the following request is valid:

```
{
  "environment" : "production",
  "scope" : {
    "nodes" : ["node1.example.com", "node2.example.com"]
  }
}
```

To run a task on a list of nodes with the `certname` value matching a regex, the following request is valid:

```
{
  "environment" : "production",
  "scope" : {
    "query" : ["from", "nodes", ["~", "certname", ".*"]]
  }
}
```

To deploy to the nodes defined by the "All Nodes" node group the following request is valid:

```
{
  "environment" : "production",
  "scope" : {
    "node_group" : "00000000-0000-4000-8000-000000000000"
  }
}
```

Targets

Targets is an array of JSON objects, each containing the following keys:

Key	Definition
hostnames	Array of hostnames that share the same target attributes. These should each match an entry in the node list scope for the job. String. This key is required.
password	Password matching the user specified. One of two credential optional, along with <code>private-key-content</code> . String.
private-key-content	Content of the ssh key used to ssh to the remote node to run on. String.
port	Specifies which port on the remote node to use to connect. Integer.
user	User on the remote system to log in as to run the task. String. This key is required.
transport	Specify ssh or winrm. String. This key is required.
run-as	An optional user to run commands when using ssh. String.
sudo-password	A password to use when changing users via <code>run-as</code> . String.
run-as-command	Command to elevate permissions via <code>run-as</code> . String.

Key	Definition
<code>connect-timeout</code>	How long, in seconds, Bolt should wait when establishing connections. Integer.
<code>tty</code>	Determines if Bolt uses pseudo tty to meet sudoer restrictions. Boolean.
<code>tmpdir</code>	The directory to upload and execute temporary files on the target. String.
<code>extensions</code>	List of file extensions that are accepted for scripts or tasks. String.

For example,

```
[
  {
    "hostnames": [ "sshnode1.example.com", "sshnode2.example.com" ],
    "password": "p@ssw0rd",
    "port": 4444,
    "user": "foo",
    "transport": "ssh"
  },
  {
    "hostnames": [ "winrmnode.example.com" ],
    "password": "p@ssw0rd!",
    "port": 4444,
    "user": "foo",
    "transport": "winrm"
  }
]
```

Response format

If the task is starts successfully, the response will have a 202 status.

The response will be a JSON object containing a link to retrieve information about the status of the job. The keys of this object are:

- `id`: an absolute URL that links to the newly created job.
- `name`: the name of the newly created job. For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
    "name" : "1234"
  }
}
```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

Key	Definition
<code>puppetlabs.orchestrator/unknown-environment</code>	If the environment does not exist, the server returns a 404 response.

Key	Definition
<code>puppetlabs.orchestrator/empty-target</code>	If the application instance specified to deploy does not exist or is empty, the server returns a 400 response.
<code>puppetlabs.orchestrator/puppetdb-error</code>	If the orchestrator is unable to make a query to PuppetDB, the server returns a 400 response.
<code>puppetlabs.orchestrator/query-error</code>	If a user does not have appropriate permissions to run a query, or if the query is invalid, the server returns a 400 response.
<code>puppetlabs.orchestrator/not-permitted</code>	This error occurs when a user does not have permission to run the task on the requested nodes. Server returns a 403 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

POST /command/task_target

Create a new task-target, which is a collection of tasks, nodes and node groups that define a permission group. When a user has permissions to run via a given task-target, they are granted permissions to run the given set of tasks, on the set of nodes described by the task-target.

After creating a task-target, use the [POST /roles](#) on page 236 endpoint to create the role that controls the permissions to execute the task-target. For the complete task-target workflow, see the blog post [Puppet Enterprise RBAC API, or how to manage access to tasks](#).

Request format

The request body must be a JSON object and uses the following keys:

Key	Definition
<code>display_name</code>	A required string that is used to name the task-target. No uniqueness requirements
<code>tasks</code>	An optional array of tasks that correspond to the task-target. There is no requirement that the tasks correspond to existing tasks. If omitted or empty, <code>all_tasks</code> must be provided and set to true.
<code>all_tasks</code>	An optional boolean to determine if all tasks can be run on designated targets. Defaults to false. If omitted, <code>tasks</code> are required to be specified. If both <code>all_tasks</code> is true and valid tasks are supplied, <code>tasks</code> are ignored and <code>all_tasks</code> is set.
<code>nodes</code>	An array of certnames that can correspond to both agent and agentless nodes. The user can run the task against the specified nodes. There is no requirement that the nodes correspond to currently available nodes. The array can be empty.
<code>node_groups</code>	An array of node-group-ids that describe the set of nodes the task can be run against. The possible nodes described by the node group will be combined with any nodes in the <code>nodes`</code> array. The array can be empty.
<code>pql_query</code>	An optional string that is a single PQL query to fetch nodes that the task-target corresponds to. The query results must contain the certnames key in the results to identify the permitted nodes.

For example:

```
{
  "display_name": "foo",
  "tasks": ["abc", "def"],
  "nodes": ["node1" "node2"],
  "node_groups": [ "000000000-0000-4000-8000-0000000000000" ]
}

{
  "display_name": "foo",
  "all_tasks": "true",
  "nodes": ["node1" "node2"],
  "node_groups": [ "000000000-0000-4000-8000-0000000000000" ]
}
```

Response format

If the task-target is successfully created, the server will return a 200 response with a JSON object containing a link to retrieve information about the task-target. The keys of this object are:

- `id`: an absolute URL to retrieve the individual task-target.
- `name`: a unique identifier for the task-target.

For example:

```
{
  "task_target": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/1",
    "name": "1"
  }
}
```

Error responses

For this endpoint, the `kind` key of the error displays the conflict. See the [Puppet orchestrator API: error responses](#) on page 611 documentation for the general format of error responses.

Key	Definition
<code>puppetlabs.orchestrator/validation-error</code>	If there is no display name supplied, there are no tasks specified when supplied, task names are not strings, <code>all_tasks</code> is not a boolean when supplied, <code>all_tasks</code> is not set to true when tasks are not supplied, node names are not strings, node-group ids are not strings, or <code>pql_query</code> is not a string the service returns a 400 response.

Related information

[POST /roles](#) on page 236

Creates a role, and attaches to it the specified permissions and the specified users and groups. Authentication is required.

POST /command/schedule_task

Schedule a task to run at a future date and time.

Request format

The request body must be a JSON object and uses the following keys:

Key	Definition
task	The task to run on the targets. Required.
params	The parameters to pass to the task. Required.
scope	The PuppetDB query, list of nodes, or a node group ID. Application scopes are not allowed for task jobs. Required.
scheduled_time	The ISO-8601 timestamp the determines when to run the scheduled job. If timestamp is in the past, a 400 error is thrown. Required.
schedule_options	Object.
interval	Object.
units	Enum in seconds.
value	Positive integer.
description	A description of the job.
environment	The environment to load the task from. The default is <code>production</code> .
noop	Whether to run the job in no-op mode. The default is <code>false</code> .

For example, to run the `package` task on `node1.example.com`, the following request is valid:

```
{
  "environment": "test-env-1",
  "task": "package",
  "params": {
    "action": "install",
    "package": "httpd"
  },
  "scope": {
    "nodes": [
      "node1.example.com"
    ]
  },
  "scheduled_time": "2027-05-05T19:50:08Z",
  "schedule_options": {
    "interval": {
      "units": "seconds",
      "value": 86400
    }
  }
}
```

Response format

If the task schedules successfully the server returns 202.

The response is a JSON object containing a link to retrieve information about the status of the job. The following keys are available.

Key	Definition
id	An absolute URL that links to the newly created job.

Key	Definition
name	The name of the newly created job.

For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/2",
    "name" : "1234"
  }
}
```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/invalid-time	If the <code>scheduled_time</code> provided is in the past, the server returns a 400 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

POST /command/schedule_plan

Schedule a plan to run at a later time. The body consists of the same fields as `/command/plan_run` and a `scheduled_time` field.

Request format

The request body must be a JSON object. The following keys are available.

Key	Definition
plan	A required string that indicates which plan to run.
params	A required object that indicates which parameters to pass to the plan.
scheduled_time	A required ISO 8601 timestamp to indicate when to run the scheduled job. If the timestamp is in the past, a 400 error will be thrown.
description	A description of the job.
environment	The environment to load the plan from. The default is <code>production</code> .

To run the package task on `node1.example.com` node, the following request is valid.

```
{
  "environment": "test-env-1",
  "plan": "facts",
  "params": {
    "targets": "node1.example.com"
  },
}
```

```
{
  "scheduled_time": "2027-05-05T19:50:08Z"
}
```

Response format

If the plan schedules successfully, the server returns 202.

The response is a JSON object containing a link to retrieve information about the status of the job. The keys are as follows.

Key	Definition
id	An absolute URL that links to the newly created job.
name	The name of the newly created job.

For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/2",
    "name" : "1234"
  }
}
```

Error responses

See [Puppet orchestrator API: error responses](#) on page 611 for the error response format. For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/invalid-time	If the scheduled_time provided is in the past, the server returns a 400 response.

POST /command/plan_run

Run a plan via the plan executor.

Request format

The request body must be a JSON object. The following keys are available:

Key	Definition
plan_name	String, the name of the plan to run.
params	Hash, the parameters the job will use.
environment	String, environment to load the plan from. The default is production.
description	String, description of the job.
userdata	An object of arbitrary key/value data supplied to the job.

To start the canary plan, the following request is valid:

```
{
  "plan_name" : "canary",
  "description" : "Start the canary plan on node1 and node2",
  "params" : {
```

```

    "nodes" : [ "node1.example.com", "node2.example.com" ],
    "command" : "whoami",
    "canary" : 1
  }
}

```

Response format

If the plan starts successfully, status 202 is returned.

The response is a JSON object containing the generated plan job name.

For example:

```

{
  "name" : "1234"
}

```

Error responses

See the [error response documentation](#) for the general format of error response. For this endpoint, the `kind` key of the error displays the conflict.

Key	Description
<code>puppetlabs.orchestrator/validation-error</code>	If the plan name is not valid the server returns a 400 response.
<code>puppetlabs.orchestrator/not-permitted</code>	This error occurs if a user makes a request they lack permissions for. The server returns a 403 response.

POST /command/environment_plan_run

Run a plan in a specific environment.

Request format

The request body must be a JSON object. The following keys are required:

Key	Definition
<code>plan_name</code>	The name of the plan to run. String.
<code>params</code>	The parameters the job will use. Parameters can include queries and node groups by passing the <code>type</code> key as part of the hash for a given parameter. Hash.

The following keys are optional:

Key	Definition
<code>environment</code>	The environment to load the plan from. The default is <code>production</code> . String.
<code>description</code>	A description of the job. String.
<code>userdata</code>	An object of arbitrary key/value data supplied to the job.

To run `example_plan`, the following request is valid:

```

{
  "plan_name" : "example_plan",

```

```

    "description" : "Output 'message' on the targets contained in 'targets'
    and 'more targets'",
    "params": {
      "message": {
        "value": "hello"
      },
      "example_hash_param": {
        "value": {
          "value": "foo"
        }
      },
      "targets": {
        "type": "query",
        "value": "nodes[certname] { }"
      },
      "more_targets": {
        "type": "node_group",
        "value": "<uuid>"
      }
    }
  }
}

```

In `example_plan`, parameters adopt the format `{<param_name>:{"value": <param_value>}}`. Orchestrator passes `<param_value>` as the parameter's value to the plan. The orchestrator also passes `hello` as the message parameter's value to the plan. For hash parameters, like `example_hash_param`, the hash `{"value": "foo"}` is passed to the plan as the value for `example_hash_param`. The `type` key is an optional way to give the orchestrator additional information about the parameter. It is not to be confused with the parameter's type in the plan metadata.

The following are the values for `type`:

Value	Definition
query	The value key is interpreted as a PuppetDB query. Orchestrator executes the query and passes the resulting list of nodes as the parameter's value to the plan. For example, for the <code>targets</code> parameter, if <code>nodes[certname] { }</code> resolves to the list <code>["foo_node", "bar_node"]</code> , orchestrator passes <code>["foo_node", "bar_node"]</code> into the plan as the parameter's value.
node_group	The value key is interpreted as a node group UUID. Orchestrator fetches all the nodes that are in this node group and passes them as the parameter's value to the plan. For example, for the <code>more_targets</code> parameter, if <code><uuid></code> resolves to <code>["group_node", "group_node_two"]</code> , orchestrator passes that array into the plan as the parameter's value.

Response format

If the plan starts successfully, the response has a 202 status. The response is a JSON object containing the generated plan job name.

For example:

```

{
  "name" : "1234"
}

```

Error response

See the [error response documentation](#) for the general format of error response. For this endpoint, the `kind` key of the error displays the conflict.

Key	Description
<code>puppetlabs.orchestrator/validation-error</code>	If the plan name is not valid the server returns a 400 response.
<code>puppetlabs.orchestrator/not-permitted</code>	This error occurs if a user makes a request they lack permissions for. The server returns a 403 response.

Puppet orchestrator API: events endpoint

Use the `/events` endpoint to learn about events that occurred during an orchestrator job.

GET /jobs/:job-id/events

Retrieve all of the events that occurred during a given job.

Parameters

The request accepts this query parameter:

Parameter	Definition
<code>start</code>	Start the list of events with the <i>nth</i> event.

For example:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?start=1272
```

Response format

The response is a JSON object that details the events in a job, and uses these keys:

Key	Definition
<code>next-events</code>	A link to the next event in the job.
<code>items</code>	A list of all events related to the job.
<code>id</code>	The job ID.
<code>type</code>	The current status of the event. See event-types.
<code>timestamp</code>	The time when the event was created.
<code>details</code>	Information about the event.
<code>message</code>	A message about the given event.

For example:

```
{
  "next-events" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?start=1272"
  },
  "items" : [ {
    "id" : "1267",
    "type" : "node_running",
    "timestamp" : "2016-05-05T19:50:08Z",
```



```

    "details" : {
      "node" : "puppet-agent.example.com",
      "detail" : {
        "noop" : false
      }
    },
    "message" : "Started puppet run on puppet-agent.example.com ..."
  }
}

```

Event types

The response format for each event contains one of these event types, which is determined by the status of the event.

Event type	Definition
node_errored	Created when there was an error running Puppet on a node.
node_failed	Created when Puppet failed to run on a node.
node_finished	Created when puppet ran successfully on a node.
node_running	Created when Puppet starts running on a node.
node_skipped	Created when a Puppet run is skipped on a node (for example, if a dependency fails).
job_aborted	Created when a job is aborted without completing.
job_stopping	Created when a stop request is received and the job is running.
job_finished	Created when a job is no longer running. The details contain final state. This should always be the last event for a job.

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the <code>start</code> parameter or the <code>job-id</code> in the request are not integers, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If the job does not exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /plan_jobs/:job-id/events

Retrieve all of the events that occurred during a given plan job.

Parameters

The request accepts the following query parameter:

Parameter	Definition
start	The lowest database ID of an event that should be shown.

For example:

https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/events?start=1272

Response format

The response is a JSON object that details the events in a plan job. The following keys and their subkeys are used:

Key	Description
next-events	A link to the next event in the job.
id	The url of the next event.
event	The next event id.

Key	
items	A list of all events related to the job.
id	The id of the event.
type	The type of event. See event-types.
timestamp	The time when the event was created.
details	Details of the event, differs based on the type of the event. See the potential values section for more detailed information.

For example:

```
{
  "next-events" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/events?start=1272",
    "event": "1272"
  },
  "items" : [ {
    "id" : "1267",
    "type" : "task_start",
    "timestamp" : "2016-05-05T19:50:08Z",
    "details" : {
      "job-id" : "8765"
    }
  },
  {
    "id" : "1268",
    "type" : "plan_finished",
    "timestamp" : "2016-05-05T19:50:14Z",
    "details" : {
      "plan-id" : "1234",
      "result" : {
        "Plan output"
      }
    }
  }
]
}
```

```
}
```

Event types

The response format for each event will contain one of the following event types, which is determined by the status of the event.

Key	Description
task_start	Created when a task run is started.
script_start	Created when a script run as part of a plan is started.
command_start	Created when a command run as part of a plan is started.
upload_start	Created when a file upload as part of a plan is started.
wait_start	Created when a <code>wait_until_available()</code> call as part of a plan is started.
out_message	Created when <code>out::message</code> is called as part of a plan. <code>details</code> will contain a message key with a value containing the message sent with <code>out::message</code> truncated to 1,024 bytes. The full message content can be obtained using the GET /plan_jobs/:job-id/event/:event-id on page 573 endpoint.
apply_start	Created when a puppet apply run as part of a plan is started.
apply_prep_start	Created when an <code>apply_prep</code> is run as part of a plan is started.
plan_finished	Created when a plan job successfully finishes.
plan_failed	Created when a plan job fails.

"Sub" plans

If a plan contains the `run_plan()` function it will begin execution of another "sub" plan during the execution of the plan job. "Sub" plans will not receive their own plan job, they will execute as part of the original plan job. There are specific events for "sub" plans that indicate a plan within a plan has started or finished:

Key	Definition
plan_start	Created when a new plan was kicked off from the current plan job using the <code>run_plan()</code> function.
plan_end	Created when a plan started using <code>run_plan()</code> has finished. Note: <code>plan_end</code> indicates the end of a "sub" plan (i.e. the plan job has not finished yet, but the "sub" plan has), while <code>plan_finished</code> indicates the end of a plan job.

Potential values in details

task and plan actions

Includes the job-id if the event is a plan action or task (i.e. `task_start`, `command_start`, `apply_start`, etc).

```
{
  "type" : "task_start",
  "timestamp" : "2019-09-30T22:22:32Z",
  "details" : {
    "job-id" : 69
  },
  "id" : "80"
}
```

`plan_finished` or `plan_failed`

If the event type is `plan_finished` or `plan_failed`, details will include `plan-id` and `result`.

```
{
  "type" : "plan_finished",
  "timestamp" : "2019-09-30T22:22:33Z",
  "details" : {
    "result" : "plan result",
    "plan-id" : "9"
  },
  "id" : "81"
}
```

`out_message`

If the event type is `out_message`, details will include `message`.

```
{
  "type" : "out_message",
  "timestamp" : "2019-09-30T22:22:32Z",
  "details" : {
    "message" : "message sent from a plan"
  }
}
```

`plan_start` or `plan_end`

If the event type is `plan_start` or `plan_end`, details will include the plan that ran (`plan`) and `plan_end` will include `duration`.

`plan_start`:

```
{
  "type" : "plan_start",
  "timestamp" : "2019-09-30T22:22:31Z",
  "details" : {
    "plan" : "test::sub_plan"
  },
  "id" : "76"
}
```

`plan_end`:

```
{
  "type" : "plan_end",
  "timestamp" : "2019-09-30T22:22:32Z",
  "details" : {
    "plan" : "test::sub_plan",
    "duration" : 0.647551
  }
}
```

```
}
```

Error responses

See the [error response documentation](#) for the general format of error responses. For the endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the start parameter or the job-id in the request are not integers, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If the plan job does not exist, the server returns a 404 response.

GET /plan_jobs/:job-id/event/:event-id

Fetch a specific event for a job.

Response format

The response is a JSON object of the event. The following keys are available.

Key	Definition
id	ID of this event.
type	Event type.
timestamp	When the event occurred.
details	Hash of detail fields specific to the event type.

Note: Unlike the /plan_jobs/:job-id/events endpoint, no truncation of fields occurs.

For example:

```
{
  "id": "1265",
  "type": "out_message",
  "timestamp": "2016-05-05T19:50:06Z",
  "details": {
    "message": "this is an output message"
  }
}
```

Error responses

See the [error response documentation](#) for the general format of error responses. For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the job-id in the request is not an integer, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If the plan job does not exist, the server returns a 404 response.
puppetlabs.orchestrator/mismatched-job-event-id	If the event-id specified does not belong to the job-id specified, the server returns a 404 response.

Puppet orchestrator API: inventory endpoint

Use the `/inventory` endpoint to discover which nodes can be reached by the orchestrator.

GET /inventory

List all nodes that are connected to the PCP broker.

Response format

The response is a JSON object containing a list of records for connected nodes, using these keys:

Key	Definition
name	The name of the connected node.
connected	The connection status is either <code>true</code> or <code>false</code> .
broker	The PCP broker the node is connected to.
timestamp	The time when the connection was made.

For example:

```
{
  "items" : [
    {
      "name" : "foo.example.com",
      "connected" : true,
      "broker" : "pcp://broker1.example.com/server",
      "timestamp" : "2016-010-22T13:36:41.449Z"
    },
    {
      "name" : "bar.example.com",
      "connected" : true,
      "broker" : "pcp://broker2.example.com/server",
      "timestamp" : "2016-010-22T13:39:16.377Z"
    }
  ]
}
```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

The server returns a 500 response if the PCP broker can't be reached.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /inventory/:node

Return information about whether the requested node is connected to the PCP broker.

Response format

The response is a JSON object indicating whether the queried node is connected, using these keys:

Key	Definition
name	The name of the connected node.
connected	The connection status is either <code>true</code> or <code>false</code> .

Key	Definition
broker	The PCP broker the node is connected to.
timestamp	The time when the connection was made.

For example:

```
{
  "name" : "foo.example.com",
  "connected" : true,
  "broker" : "pcp://broker.example.com/server",
  "timestamp" : "2017-03-29T21:48:09.633Z"
}
```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

The server returns a 500 response if the PCP broker can't be reached.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

POST /inventory

Check if the given list of nodes is connected to the PCP broker.

Request format

The request body is a JSON object specifying a list of nodes to check. For example:

```
{
  "nodes" : [
    "foo.example.com",
    "bar.example.com",
    "baz.example.com"
  ]
}
```

Response format

The response is a JSON object with a record for each node in the request, using these keys:

Key	Definition
name	The name of the connected node.
connected	The connection status is either <code>true</code> or <code>false</code> .
broker	The PCP broker the node is connected to.
timestamp	The time when the connection was made.

For example:

```
{
  "items" : [
    {
      "name" : "foo.example.com",
```

```

    "connected" : true,
    "broker" : "pcp://broker.example.com/server",
    "timestamp" : "2017-07-14T15:57:33.640Z"
  },
  {
    "name" : "bar.example.com",
    "connected" : false
  },
  {
    "name" : "baz.example.com",
    "connected" : true,
    "broker" : "pcp://broker.example.com/server",
    "timestamp" : "2017-07-14T15:41:19.242Z"
  }
]
}

```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

The server returns a 500 response if the PCP broker can't be reached.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: jobs endpoint

Use the `/jobs` endpoint to examine orchestrator jobs and their details.

GET /jobs

List all of the jobs known to the orchestrator.

Parameters

The request accepts the following query parameters:

Parameter	Definition
<code>limit</code>	Return only the most recent <i>n</i> number of jobs.
<code>offset</code>	Return results offset <i>n</i> records into the result set.
<code>order_by</code>	Return results ordered by a column. Ordering is available on <code>owner</code> , <code>timestamp</code> , <code>environment</code> , <code>name</code> , and <code>state</code> . Orderings requested on <code>owner</code> are applied to the <code>login</code> subfield of <code>owner</code> .
<code>order</code>	Indicates whether results are returned in ascending or descending order. Valid values are <code>"asc"</code> and <code>"desc"</code> . Defaults to <code>"asc"</code>
<code>type</code>	Indicates the type of job. Matches the command that created the job, which is either <code>deploy</code> , <code>task</code> , or <code>plan_task</code>
<code>task</code>	An optional task name to match against. Partial matches are supported. If <code>type</code> is specified, the <code>task</code> option is only allowed for <code>task</code> and <code>plan_task</code> types.

For example:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/jobs?
limit=20&offset=20
```

Response format

The response is a JSON object that lists orchestrator jobs and associated details and pagination information.

For information about jobs, the following keys are used:

Key	Definition
items	Contains an array of all the known jobs.
id	An absolute URL to the given job.
name	The name of the given job.
state	The current state of the job: <code>new</code> , <code>ready</code> , <code>running</code> , <code>stopping</code> , <code>stopped</code> , <code>finished</code> , or <code>failed</code> .
command	The command that created that job.
type	The type of job. Either <code>deploy</code> , <code>task</code> , <code>plan_task</code> , <code>plan_script</code> , <code>plan_upload</code> , <code>plan_command</code> , <code>plan_wait</code> , <code>plan_apply</code> , <code>plan_apply_prep</code>
options	All of the options used to create that job. The schema of options might vary based on the command.
owner	The subject id and login for the user that requested the job.
description	(deprecated) A user-provided description of the job. For future compatibility, use the description in <code>options</code> .
timestamp	The time when the job state last changed.
created_timestamp	The time the job was created.
finished_timestamp	The time the job finished.
duration	The number of seconds the job took to run.
environment	(deprecated) The environment that the job operates in.
node_count	The number of nodes the job runs on.
node_states	A JSON map containing the counts of nodes involved with the job by current node state. Unrepresented states are not displayed. This field is <code>null</code> when no nodes exist for a job.
nodes	A link to get more information about the nodes participating in a given job.
report	A link to the report for a given job.
events	A link to the events for a given job
userdata	An object of arbitrary key/value data supplied to the job.

For information about pagination, the following keys are used:

Key	Definition
pagination	Contains information about the limit, offset, and total number of items.
limit	The number of items the request was limited to.
offset	The offset from the start of the collection. Offset begins at zero.
order_by	Either owner, timestamp, environment, name, or state.
order	Either asc or desc.
total	The total number of items in the collection, ignoring limit and offset
type	The type of job filtered on, if any. Either deploy, task, or plan_task.

For example:

```
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1234",
      "name": "1234",
      "state": "finished",
      "command": "deploy",
      "type": "deploy",
      "node_count": 5,
      "node_states": {
        "finished": 2,
        "errored": 1,
        "failed": 1,
        "running": 1
      },
      "options": {
        "concurrency": null,
        "noop": false,
        "trace": false,
        "debug": false,
        "scope": {},
        "enforce_environment": true,
        "environment": "production",
        "evaltrace": false,
        "target": null,
        "description": "deploy the web app"
      },
      "owner" : {
        "email" : "admin@example.com",
        "is_revoked" : false,
        "last_login" : "2020-05-05T14:03:06.226Z",
        "is_remote" : true,
        "login" : "admin",
        "inherited_role_ids" : [ 2 ],
        "group_ids" : [ "9a588fd8-3daa-4fc2-a396-bf88945def1e" ],
        "is_superuser" : false,
        "id" : "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
        "role_ids" : [ 1 ],
        "display_name" : "Admin",
```

```

    "is_group" : false
  },
  "description": "deploy the web app",
  "timestamp": "2016-05-20T16:45:31Z",
  "started_timestamp": "2016-05-20T16:41:15Z",
  "finished_timestamp": "2016-05-20T16:45:31Z",
  "duration": "256.0",
  "environment": {
    "name": "production"
  },
  "report": {
    "id": "https://localhost:8143/orchestrator/v1/jobs/375/report"
  },
  "events": {
    "id": "https://localhost:8143/orchestrator/v1/jobs/375/events"
  },
  "nodes": {
    "id": "https://localhost:8143/orchestrator/v1/jobs/375/nodes"
  },
  "userdata": {
    "servicenow_ticket": "INC0011211"
  }
},
{
  "description": "",
  "report": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1235/report"
  },
  "name": "1235",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1235/events"
  },
  "command": "plan_task",
  "type": "plan_task",
  "state": "finished",
  "nodes": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1235/nodes"
  },
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
jobs/1235",
  "environment": {
    "name": ""
  },
  "options": {
    "description": "",
    "plan_job": 197,
    "noop": null,
    "task": "facts",
    "sensitive": [],
    "scheduled-job-id": null,
    "params": {},
    "scope": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
  },
  "project": {
    "project_id": "foo_id",
    "ref": "524df30f58002d30a3549c52c34a1cce29da2981"
  }
}

```

```

    "timestamp": "2020-09-14T18:00:12Z",
    "started_timestamp": "2020-09-14T17:59:05Z",
    "finished_timestamp": "2020-09-14T18:00:12Z",
    "duration": "67.34",
    "owner": {
      "email": "",
      "is_revoked": false,
      "last_login": "2020-08-05T17:54:07.045Z",
      "is_remote": false,
      "login": "admin",
      "is_superuser": true,
      "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
      "role_ids": [
        1
      ],
      "display_name": "Administrator",
      "is_group": false
    },
    "node_count": 1,
    "node_states": {
      "finished": 1
    },
    "userdata": {}
  },
  "pagination": {
    "limit": 20,
    "offset": 0,
    "order": "asc",
    "order_by": "timestamp",
    "total": 2,
    "type": ""
  }
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the <code>limit</code> parameter is not an integer, the server returns a 400 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /jobs/:job-id

List all details of a given job.

Response format

The response is a JSON object that lists all details of a given job, and uses these keys:

Key	Definition
id	An absolute URL to the given job.
name	The name of the given job.

Key	Definition
state	The current state of the job: new, ready, running, stopping, stopped, finished, or failed.
command	The command that created that job.
type	The type of job. Either deploy, task, plan_task, plan_script, plan_upload, plan_command, plan_wait, plan_apply, or plan_apply_prep
options	All of the options used to create that job.
owner	The subject id and login for the user that requested the job.
description	(deprecated) A user-provided description of the job.
timestamp	The time when the job state last changed.
environment	The environment that the job operates in.
node_count	The number of nodes the job runs on.
nodes	A link to get more information about the nodes participating in a given job.
report	A link to the report for a given job.
events	A link to the events for a given job.
status	The various enter and exit times for a given job state.
userdata	An object of arbitrary key/value data supplied to the job.

For example:

```
{
  "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
  "name" : "1234",
  "command" : "deploy",
  "type": "deploy",
  "state": "finished",
  "options" : {
    "concurrency" : null,
    "noop" : false,
    "trace" : false,
    "debug" : false,
    "scope" : {
      "application" : "Wordpress_app" },
    "enforce_environment" : true,
    "environment" : "production",
    "evaltrace" : false,
    "target" : null
  },
  "node_count" : 5,
  "owner" : {
    "email" : "admin@example.com",
    "is_revoked" : false,
    "last_login" : "2020-05-05T14:03:06.226Z",
    "is_remote" : true,
    "login" : "admin",
    "inherited_role_ids" : [ 2 ],
    "group_ids" : [ "9a588fd8-3daa-4fc2-a396-bf88945def1e" ],
    "is_superuser" : false,
```

```

    "id" : "751a8f7e-b53a-4ccd-9f4f-e93db6aa38ec",
    "role_ids" : [ 1 ],
    "display_name" : "Admin",
    "is_group" : false
  },
  "description" : "deploy the web app",
  "timestamp": "2016-05-20T16:45:31Z",
  "started_timestamp": "2016-05-20T16:41:15Z",
  "finished_timestamp": "2016-05-20T16:45:31Z",
  "duration": "256.0",
  "environment" : {
    "name" : "production"
  },
  "status" : [ {
    "state" : "new",
    "enter_time" : "2016-04-11T18:44:31Z",
    "exit_time" : "2016-04-11T18:44:31Z"
  }, {
    "state" : "ready",
    "enter_time" : "2016-04-11T18:44:31Z",
    "exit_time" : "2016-04-11T18:44:31Z"
  }, {
    "state" : "running",
    "enter_time" : "2016-04-11T18:44:31Z",
    "exit_time" : "2016-04-11T18:45:31Z"
  }, {
    "state" : "finished",
    "enter_time" : "2016-04-11T18:45:31Z",
    "exit_time" : null
  } ],
  "nodes" : { "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234/nodes" },
  "report" : { "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234/report" },
  "userdata": {}
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the job-id in the request is not an integer, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If the job does not exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /jobs/:job-id/nodes

List all of the nodes associated with a given job.

Request format

The request accepts the following optional query parameters:

Key	Definition
limit	Returns the most recent <i>n</i> number of jobs.
offset	Returns results offset <i>n</i> records into the result set.
order_by	Returns results ordered by a column. Ordering is available on <code>name</code> , <code>duration</code> , <code>state</code> , <code>start_timestamp</code> , and <code>finish_timestamp</code> .
order	Indicates whether results should be returned in ascending (<code>asc</code>) or descending (<code>desc</code>) order. Defaults to <code>asc</code> .

Response format

The response is a JSON object that details the nodes associated with a job.

`next-events` is an object containing information about where to find events about the job. It has the following keys:

Key	Definition
id	The url of the next event.
event	The next event id.

`items`: a list of all the nodes associated with the job, where each node has the following keys:

Key	Definition
timestamp	(deprecated) The time of the most recent activity on the node. Prefer <code>start_timestamp</code> and <code>finish_timestamp</code> .
start_timestamp	The time the node starting running or <code>nil</code> if the node hasn't started running or was skipped.
finish_timestamp	The time the node finished running or <code>nil</code> if the node hasn't finished running or was skipped.
duration	The duration of the puppet run in seconds if the node has finished running, the duration in seconds that has passed after the node started running if it is currently running, or <code>nil</code> if the node hasn't started running or was skipped.
state	The current state of the job.
transaction_uuid	(deprecated) The ID used to identify the nodes last report.
name	The hostname of the node.
details	information about the last event and state of a given node.
result	The result of a successful, failed, errored node-run. The schema of this varies.

For example:

```
{
  "next-events": {
```

```

    "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/3/
events?start=10",
    "event": "10"
  },
  "items" : [ {
    "timestamp" : "2015-07-13T20:37:01Z",
    "start_timestamp" : "2015-07-13T20:36:13Z",
    "finish_timestamp" : "2015-07-13T20:37:01Z",
    "duration" : 48.0,
    "state" : "state",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : {
      "message": "Message of latest event"
    },
    "result": {
      "output_1": "success",
      "output_2": [1, 1, 2, 3,]
    }
  }, {
    ...
  } ]
}

```

Results

The result field is available after a node finishes, fails, or errors and contains the contents of the details of the corresponding event. In task jobs this is the result of executing the task. For puppet jobs it contains metrics from the puppet run.

For example:

An error when running a task:

```

"result" : {
  "msg" : "Running tasks is not supported for agents older than version
5.1.0",
  "kind" : "puppetlabs.orchestrator/execution-failure",
  "details" : {
    "node" : "copper-6"
  }
}

```

Raw stdout from a task:

```

"result" : {
  "output" : "test\n"
}

```

Structured output from a task:

```

"result" : {
  "status" : "up to date",
  "version" : "5.0.0.201.g879fc5a-1.el7"
}

```

Error output from a task:

```

"result" : {
  "error" : "Invalid task name 'package::status'"
}

```


Puppet run results:

```
"result" : {
  "hash" : "d7ec44e176bb4b2e8a816157ebbae23b065b68cc",
  "noop" : {
    "noop" : false,
    "no_noop" : false
  },
  "status" : "unchanged",
  "metrics" : {
    "corrective_change" : 0,
    "out_of_sync" : 0,
    "restarted" : 0,
    "skipped" : 0,
    "total" : 347,
    "changed" : 0,
    "scheduled" : 0,
    "failed_to_restart" : 0,
    "failed" : 0
  },
  "environment" : "production",
  "configuration_version" : "1502024081"
}
```

Using puppet apply as part of a plan results:

```
"result" : {
  "noop": false,
  "status" : "unchanged",
  "metrics" : {
    "corrective_change" : 0,
    "out_of_sync" : 0,
    "restarted" : 0,
    "skipped" : 0,
    "total" : 347,
    "changed" : 0,
    "scheduled" : 0,
    "failed_to_restart" : 0,
    "failed" : 0
  }
}
```

Details

The details field contains information based on the last event and current state of the node and can be empty. In some cases it can duplicate data from the results key for historical reasons.

If the node state is finished or failed the details hash can include a message and a report-url. (deprecated) for jobs started with the run command it also duplicates some information from the result.

```
{
  "items" : [ {
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "finished",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : {
      "report-url" : "https://peconsole.example.com/#/cm/report/
a15bf509dd7c40705e4e1c24d0935e2e8a1591df",
      "message": "Finished puppet run on wss6c3w9wngpycg.example.com -
Success!"
    }
  }
]
```

```

    "result" : {
      "metrics" : {
        "total" : 82,
        "failed" : 0,
        "changed" : 51,
        "skipped" : 0,
        "restarted" : 2,
        "scheduled" : 0,
        "out_of_sync" : 51,
        "failed_to_restart" : 0
      },
    }, {
      ...
    } ]
  }

```

If the node state is skipped or errored, the service includes a `:detail` key that explains why a node is in that state.

```

{
  "items" : [ {
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "failed",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : {
      "message": "Error running puppet on wss6c3w9wngpycg.example.com:
java.net.Exception: Something went wrong"
    }
  }, {
    ...
  } ]
}

```

If the node state is running, the service returns the `run-time` (in seconds).

```

{
  "items" : [ {
    "timestamp" : "2015-07-13T20:37:01Z",
    "state" : "running",
    "transaction_uuid" : :uuid,
    "name" : "wss6c3w9wngpycg.example.com",
    "details" : {
      "run-time": 30,
      "message": "Started puppet run on wss6c3w9wngpycg.example.com..."
    }
  }, {
    ...
  } ]
}

```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

Key	Definition
<code>puppetlabs.orchestrator/validation-error</code>	If the <code>job-id</code> in the request is not an integer, the server returns a 400 response.
<code>puppetlabs.orchestrator/unknown-job</code>	If the job does not exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /jobs/:job-id/report

Returns a report for a given job.

Response format

The response is a JSON object that reports the status of a job, and uses these keys:

Key	Definition
items	An array of all the reports associated with a given job.
node	The hostname of a node.
state	The current state of the job.
timestamp	The time when the job was created.
events	Any events associated with that node during the job.

For example:

```
{
  "items" : [ {
    "node" : "wss6c3w9wngpycg.example.com",
    "state" : "running",
    "timestamp" : "2015-07-13T20:37:01Z",
    "events" : [ ]
  }, {
    ...
  } ]
}
```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the job-id in the request is not an integer, the server returns a 400 response.
hpuppetlabs.orchestrator/unknown-job	If the job does not exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: scheduled jobs endpoint

Use the /scheduled_jobs endpoint to gather information about orchestrator jobs scheduled to run.

GET /scheduled_jobs

List scheduled jobs in ascending order.

Parameters

The request accepts the following query parameters:

Parameter	Definition
limit	Return only the most recent <i>n</i> number of jobs.
offset	Return results offset <i>n</i> records into the result set.
order_by	One of next_run (default), scheduled_time, name.
order	asc (default) or desc
type	The type of job. Matches the command that created the job, one of: deploy, task, or plan.

For example:

```
https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs?
limit=20&offset=20
```

Response format

The response is a JSON object that contains a list of the known jobs and information about the pagination.

Key	Definition
items	Contains an array of all the scheduled jobs.
id	An absolute URL to the given job.
name	The ID of the scheduled job
type	The type of scheduled job (currently only task)
task	The name of the task associated with the scheduled task job
plan	The name of the plan associated with the scheduled plan job
scope	The specification of the targets for the task.
environment	The environment that the job operates in.
owner	The specification for the user that requested the job.
email	String
login	String
display_name	String
is_revoked	Boolean
last_login	ISO-8601 timestamp
is_remote	A boolean for the type of user
is_group	A boolean for the type of user
is_superuser	A boolean for the type of user

Key	Definition
role_ids	Array
inherited_role_ids	Array
group_ids	Array
description	A user-provided description of the job.
next_run	An ISO-8601 timestamp for the next run of a scheduled job.
scheduled_time	An ISO8601 timestamp for when the scheduled job runs.
schedule_options	Object
interval	Object
units	Enum [seconds]
value	Positive integer
noop	Boolean. Is <code>true</code> if the job runs in no-operation mode, <code>false</code> otherwise. Always <code>false</code> for task or plan jobs.
job_options	Object for the options supplied for the job.
pagination	Contains the information about the limit, offset and total number of items.
limit	A restricted number of items for the request to return.
offset	A number offset from the start of the collection (zero based).
total	The total number of items in the collection, ignoring limit and offset.
order_by	The sort field, one of <code>next_run</code> (default), <code>scheduled_time</code> , or <code>name</code> .
order	<code>asc</code> or <code>desc</code>
type	The type of job filtered on, if any. One of <code>deploy</code> , <code>task</code> , or <code>plan</code> .

For example:

```
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/1",
      "name": "1",
      "type": "task",
      "task": "echo",
      "scope": {
        "nodes": [
          "east.example.com",
          "west.example.com"
        ]
      },
      "enviroment": "production",
      "owner": {
        "email": "fred@example.com",
        "is_revoked": false,

```

```

    "last_login": "2020-05-08T15:57:28.444Z",
    "is_remote": true,
    "login": "fred",
    "inherited_role_ids": [
      2
    ],
    "group_ids": [
      "9a588fd8-3daa-4fc2-a396-bf88945def1e"
    ],
    "is_superuser": false,
    "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
    "role_ids": [
      1
    ],
    "display_name": "Fred",
    "is_group": false
  },
  "description": "rear face the cranfitouser",
  "next_run": "2018-10-12T19:50:08Z",
  "scheduled_time": "2018-10-05T19:50:08Z",
  "schedule_options": {
    "interval": {
      "units": "seconds",
      "value": 604800
    }
  },
  "noop": false,
  "job_options": {
    "noop": false,
    "task": "echo"
  }
},
{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/2",
  "name": "2",
  "type": "deploy",
  "scope": {
    "nodes": [
      "east.example.com",
      "west.example.com"
    ]
  },
  "enviroment": "production",
  "owner": {
    "email": "fred@example.com",
    "is_revoked": false,
    "last_login": "2020-05-08T15:57:28.444Z",
    "is_remote": true,
    "login": "fred",
    "inherited_role_ids": [
      2
    ],
    "group_ids": [
      "9a588fd8-3daa-4fc2-a396-bf88945def1e"
    ],
    "is_superuser": false,
    "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
    "role_ids": [
      1
    ],
    "display_name": "Fred",
    "is_group": false
  },

```

```

      "description": "middle face the cranfitouser",
      "next_run": "2019-05-05T19:50:08Z",
      "scheduled_time": "2019-05-05T19:50:08Z",
      "schedule_options": {},
      "noop": false,
      "job_options": {
        "noop": false,
        "debug": false,
        "trace": false,
        "no_noop": false,
        "evaltrace": false,
        "concurrency": null,
        "enforce_environment": false
      }
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/3",
      "name": "3",
      "type": "task",
      "task": "facter_task",
      "scope": {
        "query": "inventory[certname] { facts.aio_agent_version ~ '\\\\\\d+
\\\" }"
      },
      "environment": "production",
      "owner": {
        "email": "fred@example.com",
        "is_revoked": false,
        "last_login": "2020-05-08T15:57:28.444Z",
        "is_remote": true,
        "login": "fred",
        "inherited_role_ids": [
          2
        ],
        "group_ids": [
          "9a588fd8-3daa-4fc2-a396-bf88945def1e"
        ],
        "is_superuser": false,
        "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
        "role_ids": [
          1
        ],
        "display_name": "Fred",
        "is_group": false
      },
      "description": "front face the nebaclouser",
      "next_run": "2027-05-05T19:50:08Z",
      "scheduled_time": "2027-05-05T19:50:08Z",
      "schedule_options": {
        "interval": {
          "units": "seconds",
          "value": 86400
        }
      },
      "noop": true,
      "job_options": {
        "noop": true,
        "task": "facter_task"
      }
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/2",

```

```

    "name": "4",
    "type": "plan",
    "plan": "canary::random",
    "scope": {},
    "environment": "production",
    "owner": {
      "email": "fred@example.com",
      "is_revoked": false,
      "last_login": "2020-05-08T15:57:28.444Z",
      "is_remote": true,
      "login": "fred",
      "inherited_role_ids": [
        2
      ],
      "group_ids": [
        "9a588fd8-3daa-4fc2-a396-bf88945def1e"
      ],
      "is_superuser": false,
      "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
      "role_ids": [
        1
      ],
      "display_name": "Fred",
      "is_group": false
    },
    "description": "a fine plan",
    "next_run": "2019-05-05T19:50:08Z",
    "scheduled_time": "2019-05-05T19:50:08Z",
    "schedule_options": {},
    "noop": false,
    "job_options": {
      "noop": false,
      "plan": "canary::random",
    }
  },
],
"pagination": {
  "limit": 50,
  "offset": 0,
  "total": 4,
  "order_by": "next_run",
  "order": "asc"
}
}

```

Error responses

See [Puppet orchestrator API: error responses](#) on page 611 for the format of error responses. For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the limit or offset parameter is not an integer, the server returns a 400 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

DELETE /scheduled_jobs/:job-id

Delete a scheduled job.

- Response 204
- Response 403
 - Body

```
{
  "kind": "puppetlabs.orchestrator/not-permitted",
  "msg": "Not authorized to delete job {id}"
}
```

Puppet orchestrator API: plans endpoint

Use the /plans endpoint to see all known plans in your environments.

GET /plans

List all known plans in a given environment.

Parameters

The request accepts the following query parameters:

Parameter	Definition
environment	Return the plans in a particular environment. Defaults to production.

Response format

The response is a JSON object that lists the known plans and where to find more information about them. It uses the following keys:

Key	Definition
items	Contains an array of all known plans for the specified environment.
environment	A map containing the name key with the environment name and a code_id key indicating the code id the plans are listed from. Note: code_id is always null.

Each item above has the following keys:

Key	Definition
id	An absolute URL to retrieve plan details.
name	The full name of the plan.
permitted	A boolean indicating if the user making the request is permitted to use the plan.

For example:

```
{
```

```

    "environment": {
      "name": "production",
      "code_id": null
    },
    "items": [
      {
        "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/
profile/firewall",
        "name": "profile::firewall",
        "permitted": true
      },
      {
        "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/
profile/rolling_update",
        "name": "profile::rolling_update",
        "permitted": true
      },
      {
        "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/
canary/random",
        "name": "canary::random",
        "permitted": false
      }
    ]
  }
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the environment parameter is not a legal environment name, the server returns a 400 response.
puppetlabs.orchestrator/unknown-environment	If the specified environment doesn't exist, the server returns a 404 response.

GET /plans/:module/:planname

Return data about the specified plan, including metadata.

Parameters

Parameter	Definition
environment	Return the plan from a particular environment. Defaults to production. Note: code_id is always null.

Response format

The response is a JSON object that includes information about the specified plan. The following keys are used:

Key	Definition
id	An absolute URL to retrieve plan details.
name	The full name of the plan.

Key	Definition
environment	A map containing a name key with the environment name and a code_id key indicating the code id the plan is being listed from.
permitted	A boolean indicating if the user is permitted to use the plan or not.
metadata	A map containing a description field with the plan's description, as well as a parameters map where keys are parameter names and values that map to type, default_value, and description.
type	The type of the parameter, matches a puppet type. If no type is present for the parameter, defaults to <code>{Any`</code>
default_value	The default value of the parameter when it is not set. Optional.
description	A description of the parameter. Optional.

For example:

```
{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/plans/
package/install",
  "name": "canary::random",
  "environment": {
    "name": "production",
    "code_id": null
  },
  "metadata": {},
  "permitted": true
}
```

Error responses

Key	Definition
puppetlabs.orchestrator/validation-error	If the environment parameter is not a legal environment name, or the module or plan name is invalid, the server returns a 400 response.
puppetlabs.orchestrator/unknown-environment	If the specified environment doesn't exist, the server returns a 404 response.
puppetlabs.orchestrator/unknown-plan	If the specified plan doesn't exist within that environment, the server returns a 404 response.

Puppet orchestrator API: plan jobs endpoint

Use the `/plan_jobs` endpoint to view details about plan jobs you have run.

GET /plan_jobs

List the known plan jobs sorted by name and in descending order.

Parameters

The request accepts the following query parameters:

Parameter	Definition
limit	Return only the most recent <i>n</i> number of jobs.
offset	Return results offset <i>n</i> records into the result set.
results	Whether to include or exclude the plan output for each plan in the list. The default is include

Response format

The response is a JSON object that contains a list of the known plan jobs, and information about the pagination.

Key	Definition
items	An array of all the plan jobs.
id	An absolute URL to the given plan job.
name	The ID of the plan job.
state	The current state of the plan job: running, success, or failure
options	Information about the plan job: description, plan_name, parameters, scheduled_job_id, and environment.
description	The user-provided description for the plan job.
plan_name	The name of the plan that was run, for example <code>package::install</code> .
parameters	The parameters passed to the plan for the job.
scheduled_job_id	The <code>scheduled_job_id</code> if the plan was run as a scheduled job.
environment	The environment the plan was run in.
result	The output from the plan job.
owner	The subject ID and login for the user that requested the job.
created_timestamp	The time the plan job was created.
finished_timestamp	The time the plan job finished.
duration	The number of seconds the plan job has been running, or total duration of the plan run.
events	A link to the events for a given plan job.
status	A hash of jobs that ran as part of the plan job, with associated lists of states and their enter and exit times.
userdata	An object of arbitrary key/value data supplied to the job.
pagination	Contains the information about the limit, offset and total number of items.
limit	The number of items the request was limited to.
offset	The offset from the start of the collection (zero based).
total	The total number of items in the collection, ignoring limit and offset.

For example:

```
{
  "items": [
    {
      "finished_timestamp": "2020-09-23T18:00:13Z",
      "name": "38",
      "events": {
        "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/38/events"
      },
      "state": "success",
      "result": [
        "orchestrator.example.com: CentOS 7.2.1511 (RedHat)"
      ],
      "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/38",
      "created_timestamp": "2020-09-23T18:00:08Z",
      "duration": 123.456,
      "options": {
        "description": "just the facts",
        "plan_name": "facts::info",
        "parameters": {
          "targets": "orchestrator.example.com"
        },
        "sensitive": [],
        "scheduled_job_id": "116",
        "project": {
          "project_id": "myproject_id",
          "ref": "524df30f58002d30a3549c52c34a1cce29da2981"
        }
      },
      "owner": {
        "email": "",
        "is_revoked": false,
        "last_login": "2020-08-05T17:54:07.045Z",
        "is_remote": false,
        "login": "admin",
        "is_superuser": true,
        "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
        "role_ids": [
          1
        ],
        "display_name": "Administrator",
        "is_group": false
      },
      "userdata": {
        "servicenow_ticket": "INC0011211"
      }
    },
    {
      "finished_timestamp": null,
      "name": "37",
      "events": {
        "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/37/events"
      },
      "state": "running",
      "id": "https://orchestrator.example.com:8143:8143/orchestrator/v1/plan_jobs/37",
      "created_timestamp": "2018-06-06T20:22:08Z",
      "duration": 123.456,
      "options": {
```

```

    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
    "sensitive": ["secret"],
    "environment": "production",
    "scheduled_job_id": "5"
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false
  },
  "result": null,
  "userdata": {}
},
{
  "finished_timestamp": null,
  "name": "36",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/36/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/36",
  "created_timestamp": "2018-06-06T20:22:08Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
    "sensitive": [],
    "environment": "production",
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",

```

```

    "is_group": false
  },
  "result": null,
  "userdata": {}
},
{
  "finished_timestamp": null,
  "name": "35",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/35/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/35",
  "created_timestamp": "2018-06-06T20:22:07Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
    "sensitive": [],
    "environment": "dev",
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false
  },
  "result": null,
  "userdata": {}
},
{
  "finished_timestamp": null,
  "name": "34",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/34/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/34",
  "created_timestamp": "2018-06-06T20:22:07Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    }
  }
}

```

```

    ],
    "sensitive": [],
    "environment": "production",
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false
  },
  "result": null,
  "userdata": {}
},
{
  "finished_timestamp": null,
  "name": "33",
  "events": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/33/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/33",
  "created_timestamp": "2018-06-06T20:22:07Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    },
    "sensitive": [],
    "environment": "production",
    "scheduled_job_id": null
  },
  "owner": {
    "email": "",
    "is_revoked": false,
    "last_login": "2018-06-06T20:22:06.327Z",
    "is_remote": false,
    "login": "admin",
    "is_superuser": true,
    "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
    "role_ids": [
      1
    ],
    "display_name": "Administrator",
    "is_group": false
  },
  "result": null,
  "userdata": {}
}
}

```



```

    ],
    "pagination": {
      "limit": 6,
      "offset": 3,
      "total": 40
    }
  }
}

```

GET /plan_jobs/:job-id

List all the details of a given plan job.

Response format

The response is a JSON object that lists all details of a given plan job. The following keys are used:

Key	Defintion
id	An absolute URL to the given plan job.
name	The ID of the plan job.
state	The current state of the plan job: running,success, or failure
options	Information about the plan job: description, plan_name, and any parameters.
description	The user-provided description for the plan job.
plan_name	The name of the plan that was run, for example package::install.
parameters	The parameters passed to the plan for the job.
result	The output from the plan job.
owner	The subject ID and login for the user that requested the job.
timestamp	The time when the plan job state last changed.
created_timestamp	The time the plan job was created.
finished_timestamp	The time the plan job finished.
events	A link to the events for a given plan job.
status	A hash of jobs that ran as part of the plan job, with associated lists of states and their enter and exit times.
userdata	An object of arbitrary key/value data supplied to the job.

For example:

```

{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/1234",
  "name": "1234",
  "state": "success",
  "options": {
    "description": "This is a plan run",
    "plan_name": "package::install",
    "parameters": {
      "foo": "bar"
    }
  }
},

```

```

"result": {
  "output": "test\n"
},
"owner": {
  "email": "",
  "is_revoked": false,
  "last_login": "YYYY-MM-DDT17:06:48.170Z",
  "is_remote": false,
  "login": "admin",
  "is_superuser": true,
  "id": "42bf351c-f9ec-40af-84ad-e976fec7f4bd",
  "role_ids": [
    1
  ],
  "display_name": "Administrator",
  "is_group": false
},
"timestamp": "YYYY-MM-DDT16:45:31Z",
"status": {
  "1": [
    {
      "state": "running",
      "enter_time": "YYYY-MM-DDT18:44:31Z",
      "exit_time": "YYYY-MM-DDT18:45:31Z"
    },
    {
      "state": "finished",
      "enter_time": "YYYY-MM-DDT18:45:31Z",
      "exit_time": null
    }
  ],
  "2": [
    {
      "state": "running",
      "enter_time": "YYYY-MM-DDT18:44:31Z",
      "exit_time": "YYYY-MM-DDT18:45:31Z"
    },
    {
      "state": "failed",
      "enter_time": "YYYY-MM-DDT18:45:31Z",
      "exit_time": null
    }
  ]
},
"events": {
  "id": "https://localhost:8143/orchestrator/v1/plan_jobs/1234/events"
},
"userdata": {}
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the job-id in the request is not an integer, the server returns a 400 response.
puppetlabs.orchestrator/unknown-job	If the plan job does not exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: tasks endpoint

Use the `/tasks` endpoint to view details about the tasks pre-installed by PE and those you've installed.

GET /tasks

Lists all tasks in a given environment.

Parameters

The request accepts this query parameter:

Parameter	Definition
<code>environment</code>	Returns the tasks in the specified environment. If unspecified, defaults to <code>production</code> .

Response format

The response is a JSON object that lists each known task with a link to additional information, and uses these keys:

Key	Definition
<code>environment</code>	A map containing a <code>name</code> key specifying the environment's name and a <code>code_id</code> key indicating the code ID where the task is listed.
<code>items</code>	Contains an array of all known tasks.
<code>id</code>	An absolute URL where the task's details are listed.
<code>name</code>	The full name of the task.

```
{
  "environment": {
    "name": "production",
    "code_id": "urn:puppet:code-id:1:a86dal66c30f871823f9b2ea224796e834840676;production"
  },
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/install",
      "name": "package::install"
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/upgrade",
      "name": "package::upgrade"
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/exec/init",
      "name": "exec"
    }
  ]
}
```

Error responses

For this endpoint, the `kind` key of the error displays the conflict.

Key	Definition
<code>puppetlabs.orchestrator/validation-error</code>	If the <code>environment</code> parameter is not a legal environment name, the server returns a 400 response.
<code>puppetlabs.orchestrator/unknown-environment</code>	If the specified environment doesn't exist, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

GET /tasks/:module/:taskname

Returns data about a specified task, including metadata and file information. For the default task in a module, `:taskname` is `init`.

Parameters

The request accepts this query parameter:

Parameter	Definition
<code>environment</code>	Returns the tasks in the specified environment. If unspecified, defaults to <code>production</code> .

Response format

The response is a JSON object that includes information about the specified task, and uses these keys:

Key	Definition
<code>id</code>	An absolute URL where the task's details are listed.
<code>name</code>	The full name of the task.
<code>environment</code>	A map containing a <code>name</code> key specifying the environment's name and a <code>code_id</code> key indicating the code ID where the task is listed.
<code>metadata</code>	A map containing the contents of the <code><task>.json</code> file.
<code>files</code>	An array of the files in the task.
<code>filename</code>	The base name of the file.
<code>uri</code>	A map containing <code>path</code> and <code>params</code> fields to construct a URL to download the file content. The client determines which host to download the file from.
<code>sha256</code>	The SHA-256 hash of the file content, in lowercase hexadecimal form.
<code>size_bytes</code>	The size of the file content in bytes.

For example:

```
{
```

```

"id": "https://orchestrator.example.com:8143/orchestrator/v1/tasks/
package/install",
"name": "package::install",
"environment": {
  "name": "production",
  "code_id": "urn:puppet:code-
id:1:a86dal66c30f871823f9b2ea224796e834840676;production"
},
"metadata": {
  "description": "Install a package",
  "supports_noop": true,
  "input_method": "stdin",
  "parameters": {
    "name": {
      "description": "The package to install",
      "type": "String[1]"
    },
    "provider": {
      "description": "The provider to use to install the package",
      "type": "Optional[String[1]]"
    },
    "version": {
      "description": "The version of the package to install, defaults to
latest",
      "type": "Optional[String[1]]"
    }
  }
},
"files": [
  {
    "filename": "install",
    "uri": {
      "path": "/package/tasks/install",
      "params": {
        "environment": "production"
      }
    },
    "sha256":
"a9089b5b9720dca38a49db6f164cf8a053a7ea528711325dalc23de94672980f",
    "size_bytes": 693
  }
]
}

```

Error responses

For this endpoint, the kind key of the error displays the conflict.

Key	Definition
puppetlabs.orchestrator/validation-error	If the environment parameter is not a legal environment name, or the module or taskname is invalid, the server returns a 400 response.
puppetlabs.orchestrator/unknown-environment	If the specified environment doesn't exist, the server returns a 404 response.
puppetlabs.orchestrator/unknown-task	If the specified task doesn't exist within the specified environment, the server returns a 404 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: root endpoint

Use the `/orchestrator` endpoint to return metadata about the orchestrator API.

GET /orchestrator

A request to the root of the Puppet orchestrator returns metadata about the orchestrator API, along with a list of links to application management resources.

Response format

Responses use the following format:

```
{
  "info" : {
    "title" : "Application Management API (EXPERIMENTAL)",
    "description" : "Multi-purpose API for performing application management
operations",
    "warning" : "This version of the API is experimental, and might change
in backwards-incompatible ways in the future",
    "version" : "0.1",
    "license" : {
      "name" : "Puppet Enterprise License",
      "url" : "https://puppetlabs.com/puppet-enterprise-components-licenses"
    }
  },
  "status" : {
    "name" : "status",
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/status"
  },
  "collections" : [ {
    "name" : "environments",
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
environments"
  }, {
    "name" : "jobs",
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs"
  } ],
  "commands" : [ {
    "name" : "deploy",
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/command/
deploy"
  }, {
    "name" : "stop",
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/command/
stop"
  } ]
}
```

Error responses

For this endpoint, the server returns a 500 response.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: usage endpoint

Use the `/usage` endpoint to view details about the active nodes in your deployment.

GET /usage

List the daily node usage of the orchestrator and nodes that are present in PuppetDB.

Parameters

Parameter	Definition
<code>start_date</code>	Expected to be in the format YYYY-MM-DD, the first day to include in the result
<code>end_date</code>	Expected to be in the format YYYY-MM-DD, the last day to include in the result must be <code>>= start_date</code>

Response format

The response is a JSON object indicating the total number of active nodes and subtotals of the nodes with and without an agent installed.

The following keys are used:

Key	Definition
<code>items</code>	Contains an array entries from most recent to least recent.
<code>date</code>	An ISO 8601 date representing the date of the request in UTC.
<code>total_nodes</code>	The total number of nodes used since UTC midnight.
<code>nodes_with_agent</code>	The number of nodes in pdb at the time the request is made.
<code>nodes_without_agent</code>	The number of unique nodes that don't have an agent that were used since UTC midnight.
<code>corrective_agent_changes</code>	The number of corrective changes made by agent runs.
<code>intentional_agent_changes</code>	The number of intentional changes made by agent runs.
<code>nodes_affected_by_task_runs</code>	The number of tasks run (counted per node that a task runs on).
<code>nodes_affected_by_plan_runs</code>	The number of plans run (counted per node that a plan runs on).
<code>pagination</code>	An optional object that includes information about the original query.
<code>start_date</code>	The starting day for the request (if specified).
<code>end_date</code>	The last day requested (if specified).

```
{
  "items": [
    {
```

```

    "date": "2018-06-08",
    "total_nodes": 100,
    "nodes_with_agent": 95,
    "nodes_without_agent": 5
  }, {
    "date": "2018-06-07",
    "total_nodes": 100,
    "nodes_with_agent": 95,
    "nodes_without_agent": 5
  }, {
    "date": "2018-06-06",
    "total_nodes": 100,
    "nodes_with_agent": 95,
    "nodes_without_agent": 5
  }, {
    "date": "2018-06-05",
    "total_nodes": 100,
    "nodes_with_agent": 95,
    "nodes_without_agent": 5
  }
],
"pagination":{
  "start_date": "2018-06-01",
  "end_date": "2018-06-30"
}
}

```

Error Responses

For this endpoint, the kind key of the error displays the conflict.

Related information

[Puppet orchestrator API: error responses](#) on page 611

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Puppet orchestrator API: scopes endpoint

Use the scopes endpoint to retrieve resources for task constraints.

GET /scopes/task_targets

List all the known task targets (set of tasks + set of nodes + set of node groups) stored in orchestrator.

Response format

The response is a JSON object that lists the known task targets and uses the following keys:

Key	Definition
items	Contains an array of all the known task targets.

The following keys are included under items:

Key	Definition
id	An absolute URL to retrieve the individual task target.
name	A string that is a unique identifier for the task target.
display_name	An optional string value that can be used to describe the task target.
tasks	An optional array of tasks that this task target corresponds to.

Key	Definition
all_tasks	An optional boolean to determine if all tasks can be run on designated targets.
nodes	An array of certnames that this task target corresponds to. The array can be empty.
node_groups	An array of node group ids that this task target corresponds to. The array can be empty.
pql_query	(optional) A string that is a single PQL query that the task target corresponds to.

For example:

```
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/1",
      "name": "1",
      "tasks": [
        "package::install",
        "exec"
      ],
      "all_tasks": "false",
      "nodes": [
        "wss6c3w9wngpycg",
        "jjj2h5w8gpycgwn"
      ],
      "node_groups": [
        "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
        "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
      ],
      "pql_query": "nodes[certname] { catalog_environment = \"production
\" }"
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/2",
      "name": "2",
      "tasks": [
        "imaginary::task"
      ],
      "all_tasks": "false",
      "nodes": [
        "mynode"
      ],
      "node_groups": [
      ]
    },
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/3",
      "name": "3",
      "all_tasks": true,
      "nodes": [
        "xxx6c3w9wngpycg",
        "bbb2h5w8gpycgwn"
      ],
      "node_groups": [
        "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
```

```

    "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
  ]
}
]
}

```

GET /scopes/task_targets/:id

Get information about a specific task_target.

Response format

The response is a JSON object that provides the details about the task_target and uses the following keys:

Keys	Definitions
id	An absolute URL to retrieve the individual task_target.
name	A string that is a unique identifier for the task_target.
display_name	An optional string value that can be used to describe the task_target.
tasks	An optional array of tasks that this task_target corresponds to.
all_tasks	An optional boolean to determine if all tasks can be run on designated targets.
nodes	An array of certnames that this task_target corresponds to. The array can be empty.
node_groups	An array of node group ids that this task_target corresponds to. The array can be empty.
pql_query	(optional) A string that is a single PQL query that the task_target corresponds to.

For example:

```

{
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/1",
  "name": "1",
  "tasks": [
    "package::install",
    "exec"
  ],
  "all_tasks": "false",
  "nodes": [
    "wss6c3w9wngpycg",
    "jjj2h5w8gpycgwn"
  ],
  "node_groups": [
    "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
    "4932bfe7-69c4-412f-b15c-ac0a7c2883f1"
  ],
  "pql_query": "nodes[certname] { catalog_environment = \"production\" }"
}

```

Error responses

For this endpoint, the `kind` key of the error displays the conflict. See the [Puppet orchestrator API: error responses](#) on page 611 documentation for the general format of error responses.

Key	Definition
<code>:puppetlabs.orchestrator/unknown-task-target</code>	If the specified <code>id</code> doesn't exist, the server returns a 404 response.

Puppet orchestrator API: error responses

From time to time, you might encounter an error using the orchestrator API. In such cases, you receive an error response.

Every error response from the Puppet orchestrator is a JSON response. Each response is an object that contains the following keys:

Key	Definition
<code>kind</code>	The kind of error encountered.
<code>msg</code>	The message associated with the error.
<code>details</code>	A hash with more information about the error.

For example, if an environment does not exist for a given request, an error is raised similar to the following:

```
{
  "kind" : "puppetlabs.orchestrator/unknown-environment",
  "msg" : "Unknown environment doesnotexist",
  "details" : {
    "environment" : "doesnotexist"
  }
}
```

Migrating Bolt tasks and plans to PE

If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a PE environment. This lets you manage and run tasks and plans using PE and the console. PE environments and Bolt projects use the same default structure, and both use a Puppetfile to install content.

The *control repo* is a central Git repository PE fetches content from. An *environment* is a space for PE authors to write and install content, similar to a Bolt project.

There are two ways to get your Bolt content into an environment:

- Move your Bolt code to a new control repo. Do this if you have a `BoltDir`, or an [embedded project directory](#), in a repo that also contains other code that you do not wish to migrate to PE.
- Configure PE to point to the Bolt project. Do this if you have a dedicated repo for Bolt code, or a [local project directory](#), and don't want to duplicate it in PE.

Move Bolt content to a new PE repo

Move your Bolt project content out of your `BoltDir` and into a fresh PE control repo.

Before you begin

- Install PE on your machine. See [Getting started with Puppet Enterprise](#) on page 57.

- Set up your PE control repo and environments. See [Managing environments with a control repository](#) on page 615.

To move Bolt content to a repo:

1. Commit the contents of your Bolt project to a branch of your PE control repo.

Your new structure is similar to the [local project directory](#) in Bolt, for example:

```
project/
### Puppetfile
### bolt.yaml
### data
#   ### common.yaml
### inventory.yaml
### site-modules
    ### project
        ### manifests
        #   ### my_class.pp
        ### plans
        #   ### deploy.pp
        #   ### diagnose.pp
        ### tasks
            ### init.json
            ### init.py
```

2. Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment in PE.
3. Add the `modulepath` setting to the `environment.conf` file by adding the following line:

```
modulepath = modules:site-modules:$basemodulepath
```

Note: PE picks up modules only from the `modules` directory. It's important to add `site-modules` to the `modulepath` setting so it matches the defaults for your Bolt project. If you have a `modulepath` setting in `bolt.yaml`, match it to the `modulepath` setting in `environment.conf`.

4. Publish the branch to the PE control repo.
5. Deploy code using `puppet code deploy --<ENVIRONMENT>`, where `<ENVIRONMENT>` is the name of your branch, to commit the new branch to Git.

Note: You can also deploy code using a webhook. See [Triggering Code Manager with a webhook](#) on page 645 for more information.

After you deploy code, modules, and the tasks and plans within them, listed in the new environment's Puppetfile will be available to use in PE.

Related information

[Plans in PE versus Bolt plans](#) on page 518

Some plan language functions, features, and behaviors are different in PE compared to Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

Point PE to a Bolt project

Allow PE to manage content in your dedicated Bolt repo.

Before you begin

- Install PE on your machine. See [Getting started with Puppet Enterprise](#) on page 57.
- Ensure your Bolt project follows the [local project directory](#) structure.

To point PE to your Bolt content:

1. Generate a private SSH key to allow access to the control repository.

This SSH key cannot require a password.

- a) Generate the key pair:

```
ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
```

- b) Set appropriate permissions so that the `pe-puppet` user can access the key:

```
puppet infrastructure configure
```

Your keys are now located in `/etc/puppetlabs/puppetserver/ssh`:

- Private key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
- Public key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub`

Configure your Git host to use the SSH public key you generated. The process to do this is different for every Git host. Usually, you create a user or service account, and then assign the SSH public key to it.

Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

See the your Git host docs for detailed instructions on adding SSH keys to your Git server. Commonly used Git hosts include:

- [GitHub](#)
- [BitBucket Server](#)
- [GitLab](#)

2. Change the name of your branch to `production`. PE uses branches in Git as environments and the default environment is `production`.
3. Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment.
4. Add the `modulepath` setting to the `environment.conf` file by adding the following line:

```
modulepath = modules:site-modules:$basemodulepath
```

Note: Without this setting, PE picks up modules only from the `modules` directory. It's important to add `site-modules` to the `modulepath` setting so it matches the defaults for your Bolt project. If you have a `modulepath` setting in `bolt.yaml`, match it to the `modulepath` setting in `environment.conf`.

5. Publish the branch to the PE control repo.
6. Deploy code using `puppet code deploy --<ENVIRONMENT>`, where `<ENVIRONMENT>` is the name of your branch, to commit the new branch to git.

Note: You can also deploy code using a webhook. See [Triggering Code Manager with a webhook](#) on page 645 for more information.

After you deploy code, modules, and the tasks and plans within them, listed in the new environment's Puppetfile will be available to use in PE.

Related information

[Plans in PE versus Bolt plans](#) on page 518

Some plan language functions, features, and behaviors are different in PE compared to Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

PE workflows for Bolt users

Review these differences between PE and Bolt commands and workflows before you start running tasks and plans in PE.

Connecting to nodes

You must connect PE to each node you want to run tasks on or include in a plan. See [Add nodes to the inventory](#) on page 59 for instructions on how to add agent or agentless nodes to your inventory.

Installing tasks and plans

In PE, as in Bolt, you use the `mod` command to download modules. But instead of running the `bolt puppetfile install` command to install them, you trigger Code Manager and deploy code using the `puppet code deploy` command. See [Triggering Code Manager on the command line](#) on page 639 for more information.

Running tasks and plans

PE does not recognize the `bolt` command for running tasks and plans. Instead, use the `puppet task run` and `puppet plan run` commands, or use the console.

To run tasks or plans from the command line, see:

- [Running tasks from the command line](#) on page 497
- [Running plans from the command line](#) on page 521

To run tasks or plans from the console, see:

- [Running tasks from the console](#) on page 490
- [Running plans from the console](#) on page 521

Limitations in PE

Not everything in Bolt works in PE. For example, many pre-installed Bolt modules are not included in PE and many plan functions like `file::exists` and `set_feature` do not work. See [Plans in PE versus Bolt plans](#) on page 518 for more information.

Managing and deploying Puppet code

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

Code management tools use Git version control to track, maintain, and deploy your Puppet modules, manifests, and data. This allows you to more easily maintain, update, review, and deploy Puppet code and data for multiple environments.

Code Manager automates the deployment of your Puppet code and data. You make code and data changes on your workstation, push changes to your Git repository, and then Code Manager creates environments, installs modules, and deploys the new code to your primary server and compilers, without interrupting agent runs.

If you are already using r10k to manage your Puppet code, we suggest that you upgrade to Code Manager. Code Manager works in concert with r10k, so when you switch to Code Manager, you no longer interact directly with r10k.

If you're using r10k and aren't ready to switch to Code Manager yet, you can continue using r10k alone. You push your code changes to your version control repo, deploy environments from the command line, and r10k creates environments and installs the modules for each one.

Both Code Manager and the r10k code management tool are built into PE, so you don't have to install anything. To begin managing your code with either of these tools, you perform the following tasks:

1. Create a control repository with Git for maintaining your environments and code.

The control repository is the Git repository that code management uses to maintain and deploy your Puppet code and data and to create environments in your Puppet infrastructure. As you update your control repo, code management keeps each of your environments updated.

2. Set up a Puppetfile to manage content in your environment.

This file specifies which modules and data to install in your environment, including what version of that content to install, and where to download the content from.

3. Configure Code Manager (recommended) or r10k.

Configure code management in the console's primary server profile. If you need to customize your configuration further, you can do so by adding keys to Hiera.

4. Deploy environments. With Code Manager, either set up a deployment trigger (recommended), or plan to trigger your deployment from the command line, which is useful for initial configuration. If you are using r10k alone, run it from the command line whenever you want to deploy.

The following sections go into detail about setting up and using Code Manager and r10k.

- [Managing environments with a control repository](#) on page 615

To manage your Puppet code and data with either Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

- [Managing environment content with a Puppetfile](#) on page 620

A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

- [Managing code with Code Manager](#) on page 625

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

- [Managing code with r10k](#) on page 663

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository.

- [About file sync](#) on page 674

File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

Managing environments with a control repository

To manage your Puppet code and data with either Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

How the control repository works

Code management relies on version control to track, maintain, and deploy your Puppet code and data. The control repository (or repo) is the Git repository that code management uses to manage environments in your infrastructure. As you update code and data in your control repo, code management keeps each of your environments updated.

Code management creates and maintains your environments based on the branches in your control repo. For example, if your control repo has a production branch, a development branch, and a testing branch, code management creates a production environment, a development environment, and a testing environment, each with its own version of your Puppet code and data.

Environments are created in `/etc/puppetlabs/code/environments` on the primary server. To learn more about environments in Puppet, read the documentation about [environments](#).

To create a control repo that includes the standard recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations. If you cannot access the internet or cannot use modules directly from the Forge because of your organization's security rules, create an empty control repo and add the files you need to it.

Note: For Windows systems, be sure your version control is configured to use CRLF line endings. See your version control system for instructions on how to do this.

At minimum, a control repo comprises:

- A Git remote repository. The remote is where your control repo is stored on your Git host.
- A default branch named `production`, rather than the usual Git default of `master`.
- A Puppetfile to manage your environment content.
- An `environment.conf` file that modifies the `$modulepath` setting to allow environment-specific modules and settings.



CAUTION: Enabling code management means that Puppet manages the environment directories and **existing environments are not preserved**. Environments with the same name as the new one are overwritten. Environments not represented in the control repo are erased. If you were using environments before, commit any necessary files or code to the appropriate new control repo branch, or back them up somewhere *before* you start configuring code management.

Create a control repo from the Puppet template

To create a control repo that includes a standard recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

To base your control repo on the Puppet [control repository template](#), you copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

The control repo template contains the files needed to get you started with a functioning control repo, including:

- An `environment.conf` file to implement a `site-modules/` directory for roles, profiles, and custom modules.
- `config_version` scripts to notify you which control repo version was applied to the agents.
- Basic code examples for setting up roles and profiles.
- An example `hieradata` directory that matches the default hierarchy.
- A Puppetfile to manage content maintained in your environment.

Set up a private SSH key so that your primary server can identify itself to your Git host.

1. Generate a private SSH key to allow access to the control repository.

This SSH key cannot require a password.

- a) Generate the key pair:

```
ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
```

- b) Set appropriate permissions so that the `pe-puppet` user can access the key:

```
puppet infrastructure configure
```

Your keys are now located in `/etc/puppetlabs/puppetserver/ssh`:

- Private key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
- Public key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub`

Configure your Git host to use the SSH public key you generated. The process to do this is different for every Git host. Usually, you create a user or service account, and then assign the SSH public key to it.

Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

See the your Git host docs for detailed instructions on adding SSH keys to your Git server. Commonly used Git hosts include:

- [GitHub](#)
- [BitBucket Server](#)
- [GitLab](#)

2. Create a repository in your Git account, with the name you want your control repo to have.

Steps for creating a repository vary, depending on what Git host you are using (GitHub, GitLab, Bitbucket, or another provider). See your Git host's documentation for complete instructions.

For example, on GitHub:

- a) Click + at the top of the page, and choose **New repository**.
- b) Select the account **Owner** for the repository.
- c) Name the repository (for example, `control_repo`).
- d) Note the repository's SSH URL for later use.

3. If you don't already have Git installed, run the following command on your primary server:

```
yum install git
```

4. From the command line, clone the Puppet `control_repo` template.

```
git clone https://github.com/puppetlabs/control_repo.git
```

5. Change directory into your control repo.

```
cd <NAME OF YOUR CONTROL REPO>
```

6. Remove the template repository as your default source.

```
git remote remove origin
```

7. Add the control repository you created as the default source.

```
git remote add origin <URL OF YOUR GIT REPOSITORY>
```

8. Push the contents of the cloned control repo to your remote copy of the control repo:

```
git push origin production
```

You now have a control repository based on the Puppet `control-repo` template. When you make changes to this repo on your workstation and push those changes to the remote copy of the control repo on your Git server, Code Manager deploys your infrastructure changes.

You also now have a Puppetfile available for you to start adding and managing content, like module code.

Create an empty control repo

If you can't use the control repo template because, for example, you cannot access the internet or use modules directly from the Forge because of your security rules, create an empty control repo and then add the files you need.

To start with an empty control repo, you create a new repo on your Git host and then copy it to your workstation. You make some changes to your repo, including adding a configuration file that allows code management tools to find modules in both your site and environment-specific module directories. When you're done making changes, push your changes to your repository on your Git host.

1. Generate a private SSH key to allow access to the control repository.

This SSH key cannot require a password.

- a) Generate the key pair:

```
ssh-keygen -t rsa -b 2048 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa
```

- b) Set appropriate permissions so that the `pe-puppet` user can access the key:

```
puppet infrastructure configure
```

Your keys are now located in `/etc/puppetlabs/puppetserver/ssh`:

- Private key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`
- Public key: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa.pub`

Configure your Git host to use the SSH public key you generated. The process to do this is different for every Git host. Usually, you create a user or service account, and then assign the SSH public key to it.

Code management needs read access to your control repository, as well as any module repositories referenced in the Puppetfile.

See the your Git host docs for detailed instructions on adding SSH keys to your Git server. Commonly used Git hosts include:

- [GitHub](#)
- [BitBucket Server](#)
- [GitLab](#)

2. Create a repository in your Git account, with the name you want your control repo to have.

Steps for creating a repository vary, depending on what Git host you are using (GitHub, GitLab, Bitbucket, or another provider). See your Git host's documentation for complete instructions.

For example, on GitHub:

- a) Click + at the top of the page, and choose **New repository**.
- b) Select the account **Owner** for the repository.
- c) Name the repository (for example, `control-repo`).
- d) Note the repository's SSH URL for later use.

3. Clone the empty repository to your workstation by running `git clone <REPOSITORY URL>`
4. Create a file named `environment.conf` in the main directory of your control repo.

5. In your text editor, open `environment.conf` file, and add the line below to set the modulepath. Save and close the file.

```
modulepath=site-modules:modules:$basemodulepath
```

6. Add and commit your change to the repository by running:

- a) `git add environment.conf`
- b) `git commit -m "add environment.conf"`

The `environment.conf` file allows code management tools to find modules in both your site and environment-specific module directories. For details about this file, see the [environment.conf](#) documentation.

7. Rename the master branch to production by running `git branch -m master production`

Important: The default branch of a control repo must be production.

8. Push your repository's production branch from your workstation to your Git host by running `git push -u origin production`

When you make changes to this repo on your workstation, push those changes to the remote copy of the control repo on your Git server, so that code management can deploy your infrastructure changes.

After you've set up your control repo, create a Puppetfile for managing your environment content with code management.

If you already have a Puppetfile, you can now configure code management. Code management configuration steps differ, depending on whether you're using Code Manager (recommended) or r10k. For important information about the function and limitations of each of these tools, along with configuration instructions, see the Code Manager and r10k pages.

Add an environment

Create new environments by creating branches based on the production branch of your control repository.

Before you begin

Make sure you have:

- Configured either [Code Manager](#) or [r10k](#).
- Created a [Puppetfile](#) in the default (usually 'production') branch of your control repo.
- Selected your code management deployment method (such as the `puppet-code` command or a [webhook](#)).

Remember: If you are using multiple control repos, do not duplicate branch names unless you use a source prefix. For more information about source prefixes, see the documentation about configuring [sources](#).

1. Create a new branch: `git branch <NAME-OF-NEW-BRANCH>`
2. Check out the new branch: `git checkout <NAME-OF-NEW-BRANCH>`
3. Edit the Puppetfile to track the modules and data needed in your new environment, and save your changes.
4. Commit your changes: `git commit -m "a commit message summarizing your change"`
5. Push your changes: `git push origin <NAME-OF-NEW-BRANCH>`
6. Deploy your environments as you normally would, either on the command line or with a Code Manager webhook.

Delete an environment with code management

To delete an environment with Code Manager or r10k, delete the corresponding branch from your control repository.

1. On the production branch of your control repo directory, on the command line, delete the environment's corresponding remote branch by running `git push origin --delete <BRANCH-TO-DELETE>`
2. Delete the local branch by running `git branch -d <BRANCH-TO-DELETE>`

3. Deploy your environments as you normally would, either on the command line or with a Code Manager webhook.

Note: If you use Code Manager to deploy environments with the webhook, deleting a branch from your control repository does not immediately delete that environment from the primary server's live code directories. Code Manager deletes the environment when it next deploys changes to any other environment. Alternately, to delete the environment immediately, deploy all environments manually: `puppet-code deploy --all --wait`

Managing environment content with a Puppetfile

A Puppetfile specifies detailed information about each environment's Puppet code and data, including where to get that code and data from, where to install it, and whether to update it.

Both Code Manager and r10k use a Puppetfile to install and manage the content of your environments.

The Puppetfile

The Puppetfile specifies the modules and data that you want in each environment. The Puppetfile can specify what version of modules you want, how the modules and data are to be loaded, and where they are placed in the environment.

A Puppetfile is a formatted text file that specifies the modules and data that you want brought into your control repo. Typically, a Puppetfile controls content such as:

- Modules from the Forge
- Modules from Git repositories
- Data from Git repositories

For each environment that you want to manage content in, you need a Puppetfile. Create a base Puppetfile in your default environment (usually `production`). As you create new branches based on your default branch, each environment inherits this base Puppetfile. You can then edit each environment's Puppetfile as needed.

Managing modules with a Puppetfile

With code management, install and manage your modules only with a Puppetfile.

Almost all Puppet manifests are kept in modules, collections of Puppet code and data with a specific directory structure. By default, Code Manager and r10k install content in a modules directory (`./modules`) in the same directory the Puppetfile is in. For example, declaring the `puppetlabs-apache` module in the Puppetfile normally installs the module into `./modules/apache`. To learn more about modules, see the [module](#) documentation.

Important:

With Code Manager and r10k, **do not** use the `puppet module` command to install or manage modules. Instead, code management depends on the Puppetfile in each of your environments to install, update, and manage your modules. If you've installed modules to the live code directory with `puppet module install`, Code Manager deletes them.

The Puppetfile does NOT include Forge module dependency resolution. You must make sure that you have every module needed for all of your specified modules to run. In addition, Forge module symlinks are unsupported; when you install modules with r10k or Code Manager, symlinks are not installed.

Including your own modules

If you develop your own modules, maintain them in version control and include them in your Puppetfile as you would declare any module from a repository. If you have content in your control repo's module directory that is *not* listed in your Puppetfile, code management purges it. (The control repo module directory is, by default, `./modules` relative to the location of the Puppetfile.)

Deploying code

When you install or update a module, you must trigger Code Manager or r10k to deploy the new or updated code to your environments.

With Code Manager, you can deploy code on the command line or using a webhook:

- [Triggering Code Manager on the command line](#) on page 639
- [Triggering Code Manager with a webhook](#) on page 645

With r10k, you can deploy code using the command line:

- [Deploying environments with r10k](#) on page 671

Creating a Puppetfile

Your Puppetfile manages the content you want to maintain in that environment.

In a Puppetfile, you can declare:

- Modules from the Forge.
- Modules from a Git repository.
- Data or other non-module content (such as Hiera data) from a Git repository.

You can declare any or all of this content as needed for each environment. Each module or repository is specified with a `mod` directive, along with the name of the content and other information the Puppetfile needs so that it can correctly install and update your modules and data.`

It's best to create your first Puppetfile in your production branch. Then, as you create branches based on your production branch, edit each branch's Puppetfile as needed.

Create a Puppetfile

Create a Puppetfile that manages the content maintained in your environment.

Before you begin

Set up a control repo, with `production` as the default branch. To learn more about control repositories, see the [control repository](#) documentation.

Create a Puppetfile in your production branch, and then edit it to declare the content in your production environment with the `mod` directive.

1. On your production branch, in the root directory, create a file named `Puppetfile`.
2. In a text editor, for example Visual Studio Code (VS Code), edit the Puppetfile, declaring modules and data content for your environment. Note that Puppet has an [extension](#) for VS Code that supports syntax highlighting of the Puppet language.

You can declare modules from the Forge or you can declare Git repositories in your Puppetfile. See the related topics about declaring content in the Puppetfile for details and code examples.

Configure Code Manager or r10k.

Change the Puppetfile module installation directory

If needed, you can change the directory to which the Puppetfile installs all modules.

To specify a module installation path other than the default modules directory (`./modules`), use the `moduledir` directive.

This directive applies to *all* content declared in the Puppetfile. You must specify this as a relative path at the top of the Puppetfile, **before** you list any modules.

To change the installation paths for only certain modules or data, declare those content sources as Git repositories and set the `install_path` option. This option overrides the `moduledir` directive. See the related topic about how to declare content as a Git repo for instructions.

Add the `moduledir` directive at the top of the Puppetfile, specifying your module installation directory relative to the location of the Puppetfile.

```
moduledir 'thirdparty'
```

Declare Forge modules in the Puppetfile

Declare Forge modules in your Puppetfile, specifying the version and whether or not code management keeps the module updated.

Specify modules by their full name. You can specify the most recent version of a module, with or without updates, or you can specify a specific version of a module.

Note: Some existing Puppetfiles contain a `forge` setting that provides legacy compatibility with `librarian-puppet`. This setting is non-operational for Code Manager and r10k. To configure how Forge modules are downloaded, specify `forge_settings` in Hiera instead. See the topics about configuring the Forge settings for Code Manager or r10k for details.

In your Puppetfile, specify the modules to install with the `mod` directive. For each module, pass the module name as a string, and optionally, specify what version of the module you want to track.

If you specify no options, code management installs the latest version and keeps the module at that version. To keep the module updated, specify `:latest`. To install a specific version of the module and keep it at that version, specify the version number as a string.

```
mod 'puppetlabs/apache'
mod 'puppetlabs/ntp', :latest
mod 'puppetlabs/stdlib', '0.10.0'
```

This example:

- Installs the latest version of the `apache` module, but does not update it.
- Installs the latest version of the `ntp` module, and updates it when environments are deployed.
- Installs version 0.10.0 of the `stdlib` module, and does not update it.

Declare Git repositories in the Puppetfile

List the modules, data, or other non-module content that you want to install from a Git repository.

To specify any environment content as a Git repository, use the `mod` directive with the `:git` option. This is useful for modules you don't get from the Forge, such as your own modules, as well as data or other non-module content.

To install content and keep it updated to the master branch, declare the content name and specify the repository with the `:git` directive. Optionally, specify an `:install_path` for the content.

```
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache'
mod 'site_data',
  :git => 'git@git.example.com:site_data.git',
  :install_path => 'hieradata'
```

This example installs the `apache` module and keeps that module updated with the master branch of the listed repository. It also installs site data content from a Git repository into the environment's `./hieradata/site_data` subdirectory.

Note: Content is installed in the modules directory and treated as a module, unless you use the `:install_path` option. Use this option with non-module content to keep your data separate from your modules.

Specify installation paths for repositories

You can set individual installation paths for any of the repositories that you declare in the Puppetfile.

The `:install_path` option allows you to separate non-module content in your directory structure or to set specific installation paths for individual modules. When you set this option for a specific repository, it overrides the `moduledir` setting.

To install the content into a subdirectory in the environment, specify the directory with the `install_path` option. To install into the root of the environment, specify an empty value.

```
mod 'site_data_1',
  :git => 'git@git.example.com:site_data_1.git',
  :install_path => 'hieradata'
mod 'site_data_2',
  :git => 'git@git.example.com:site_data_2.git',
  :install_path => ''
```

This example installs site data content from the `site_data_1` repository into `./hieradata/site_data` and content from `site_data_2` into `./site_data` subdirectory.

Declare module or data content with SSH private key authentication

To declare content protected by SSH private keys, declare the content as a repository, and then configure the private key setting in your code management tool.

1. Declare your repository content, specifying the Git repo by the SSH URL.

```
mod 'myco/privatemod',
  :git => 'git@git.example.com:myco/privatemod.git'
```

2. Configure the correct private key by setting Code Manager or `r10k` parameters in Hiera:

- To set a key for all Git operations, use the private key setting under `git-settings`.
- To set a private key for an individual remote, set the private key in the `repositories` hash in `git-settings` for each specific remote.

For more information, see [Configuring Git settings](#) on page 633.

Keep repository content at a specific version

The Puppetfile can maintain repository content at a specific version.

To specify a particular repository version, declare the version you want to track with your choice of the following options. Setting one of these options maintains the repository at that version and deploys any updates you make to that particular version.

- `ref`: Specifies the Git reference to check out. This option can reference either a tag, a commit, or a branch.
- `tag`: Specifies the repo by a certain tag value.
- `commit`: Specifies the repo by a certain commit.
- `branch`: Specifies a certain branch of the repo.
- `default_branch`: Specifies a default branch to use for deployment if the specified `ref`, `tag`, `commit`, or `branch` cannot be deployed. You must also specify one of the other version options. This is useful if you are tracking a relative branch of the control repo.

In the Puppetfile, declare the content, specifying the repository and version you want to track.

To install `puppetlabs/apache` and specify the '0.9.0' tag, use the `tag` option.

```
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache',
  :tag => '0.9.0'
```

To install puppetlabs/apache and use the branch option to track the 'proxy_match' branch.

```
mod 'apache',
  :git      => 'https://github.com/puppetlabs/puppetlabs-apache',
  :branch   => 'proxy_match'
```

To install puppetlabs/apache and use the commit option to track the '8df51aa' commit.

```
mod 'apache',
  :git      => 'https://github.com/puppetlabs/puppetlabs-apache',
  :commit   => '8df51aa'
```

Declare content from a relative control repo branch

The branch option also has a special `:control_branch` option, which allows you to deploy content from a control repo branch relative to the location of the Puppetfile.

Normally, branch tracks a specific named branch of a repo, such as testing. If you set it to `:control_branch`, it instead tracks whatever control repo branch the Puppetfile is in. For example, if your Puppetfile is in the production branch, content from the production branch is deployed; if a duplicate Puppetfile is located in testing, content from testing is deployed. This means that as you create new branches, you don't have to edit the inherited Puppetfile as extensively.

Note: The module repository branch names must match the control repository branch names in order to use `:control_branch`.

To track a branch within the control repo, declare the content with the `:branch` option set to `:control_branch`.

```
mod 'hieradata',
  :git      => 'git@git.example.com:organization/hieradata.git',
  :branch   => :control_branch
```

Set a default branch for content deployment

Set a `default_branch` option to specify what branch code management deploys content from if the given option for your repository cannot be deployed.

If you specified a ref, tag, commit, or branch option for your repository, and it cannot be resolved and deployed, code management can instead deploy the `default_branch`. This is mostly useful when you set branch to the `:control_branch` value.

If your specified content cannot be resolved and you have not set a default branch, or if the default branch cannot be resolved, code management logs an error and does not deploy or update the content.

In the Puppetfile, in the content declaration, set the `default_branch` option to the branch you want to deploy if your specified option fails.

```
# Track control branch and fall-back to master if no matching branch.
mod 'hieradata',
  :git      => 'git@git.example.com:organization/hieradata.git',
  :branch   => :control_branch,
  :default_branch => 'master'
```


Managing code with Code Manager

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

- [How Code Manager works](#) on page 625

Code Manager uses r10k and the file sync service to stage, commit, and sync your code, automatically managing your environments and modules.

- [Configure Code Manager](#) on page 626

To configure Code Manager, first enable Code Manager in Puppet Enterprise (PE), then set up authentication, and test the communication between the control repository and Code Manager.

- [Deploying Code without blocking requests to Puppet Server](#) on page 630

When compiling code, Puppet Server typically blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. You can enable *lockless deploys* so the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.

- [Customize Code Manager configuration in Hiera](#) on page 631

To customize your Code Manager configuration, set parameters with Hiera.

- [Triggering Code Manager on the command line](#) on page 639

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

- [Triggering Code Manager with a webhook](#) on page 645

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. The webhook is the simplest way to trigger Code Manager.

- [Triggering Code Manager with custom scripts](#) on page 646

Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repos, or custom notifications.

- [Troubleshooting Code Manager](#) on page 648

Code Manager requires coordination between multiple components, including r10k and the file sync service. If you have issues with Code Manager, check that these components are functioning.

- [Code Manager API](#) on page 651

Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

How Code Manager works

Code Manager uses r10k and the file sync service to stage, commit, and sync your code, automatically managing your environments and modules.

First, create a control repository with branches for each environment that you want to create (such as production, development, or testing). Create a Puppetfile for each of your environments, specifying exactly which modules to install in each environment. This allows Code Manager to create directory environments, based on the branches you've set up. When you push code to your control repo, you trigger Code Manager to pull that new code into a staging code directory (`/etc/puppetlabs/code-staging`). File sync then picks up those changes, pauses Puppet Server to avoid conflicts, and then syncs the new code to the live code directories on your primary server and compilers.

For more information about using environments in Puppet, see [About Environments](#).

Understanding file sync and the staging directory

To sync your code across your primary server and compilers, and to make sure that code stays consistent, Code Manager relies on file sync and two different code directories: the staging directory and the live code directory.

Without Code Manager or file sync, Puppet code lives in the codedir, or live code directory, at `/etc/puppetlabs/code`. But the file sync service looks for code in a code staging directory (`/etc/puppetlabs/code-staging`), and then syncs that to the live codedir.

Code Manager moves new code from source control into the staging directory, and then file sync moves it into the live code directory. This means you no longer write code to the codedir; if you do, the next time Code Manager deploys code from source control, it overwrites your changes.

For more detailed information about how file sync works, see the related topic about file sync.



CAUTION:

Don't directly modify code in the staging directory. Code Manager overwrites it with updates from the control repo. Similarly, don't modify code in the live code directory, because file sync overwrites that with code from the staging directory.

Related information

[About file sync](#) on page 674

File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

Environment isolation metadata and Code Manager

Both your live and staging code directories contain metadata files that are generated by file sync to provide environment isolation for your resource types.

These files, which have a `.pp` extension, ensure that each environment uses the correct version of the resource type. Do not delete or modify these files. Do not use expressions from these files in regular manifests.

These files are generated as Code Manager deploys new code in your environments. If you are new to Code Manager, these files are generated when you first deploy your environments. If you already use Code Manager, the files are generated as you make and deploy changes to your existing environments.

For more details about these files and how they isolate resource types in multiple environments, see [environment isolation](#).

Moving from r10k to Code Manager

Switching from r10k to Code Manager can improve automation of your code management and deployments, but some r10k users might not be ready to switch yet.

Code Manager uses r10k in the background to improve automation of your code management and deployment. However, switching to Code Manager means you can no longer use r10k manually. Because of this, some r10k users might not want to move to Code Manager yet. Specifically, be aware of the following restrictions:

- If you use Code Manager, you **cannot** deploy code manually with r10k. If you depend on the ability to deploy modules directly from r10k, using the `r10k deploy module` command, continue to use your current r10k workflow.
- Code Manager must deploy all control repos to the same directory. If you are using r10k to deploy control repos to different directories, continue to use your current r10k workflow.
- Code Manager does not support system `.SSH` configuration or other shellgit options.
- Code Manager does not support post-deploy scripts.

If you rely on any of the above configurations, or any other r10k configuration that Code Manager doesn't yet support, continue to use your current r10k workflow. If you are using a custom script to deploy code, carefully assess whether Code Manager meets your needs.

Related information

[Managing code with r10k](#) on page 663

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository.

Configure Code Manager

To configure Code Manager, first enable Code Manager in Puppet Enterprise (PE), then set up authentication, and test the communication between the control repository and Code Manager.

Complete the following steps to enable and configure Code Manager.

1. [Upgrade from r10k to Code Manager](#) on page 627

2. [Enable Code Manager](#) on page 627
3. [Set up authentication for Code Manager](#) on page 63
4. [Test the control repo](#) on page 628
5. [Test Code Manager](#) on page 629

Related concepts

[Deploying Code without blocking requests to Puppet Server](#) on page 630

When compiling code, Puppet Server typically blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. You can enable *lockless deploys* so the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.

Upgrade from r10k to Code Manager

To upgrade from r10k to Code Manager, you must disable the previous r10k installation.

You must complete the following pre-requisites when you upgrade from r10k to Code Manager:

If you used any previous versions of r10k:

Disable any tools that may automatically run it. Most commonly, this is the `zack-r10k` module.

Important: Code Manager cannot properly install or update code when other tools are running r10k.

When you use Code Manager, it runs r10k in the background. You cannot directly interact with r10k or use the `zack-r10k` module.

Enable Code Manager

Enable Code Manager to connect your primary server to your Git repository.

Before you begin

You must have an SSH key without a passphrase and is saved on your primary server at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`.

1. In the console, in the **PE Master** node group, set parameters for the `puppet_enterprise::profile::master` class.
 - `code_manager_auto_configure` - Specify true to enable Code Manager.
 - `r10k_remote` - Enter a string that is a valid SSH URL for your Git control repository. For example: `"git@<YOUR.GIT.SERVER.COM>:puppet/control.git"`.

Note: Some Git providers, such as Bitbucket, may have additional requirements for enabling SSH access. See your provider's documentation for information.

 - `r10k_private_key` - Enter `" /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"`. This is the path to the private key that permits the `pe-puppet` user access to your Git repositories.
2. Run Puppet on your primary server and all compilers.

Note:

If you run Puppet for your primary server and all compilers at the same time, such as with **Run Puppet** in the console, the following errors might display in your compilers' logs:

```
2015-11-20 08:14:38,308 ERROR [clojure-agent-send-off-pool-0]
[p.e.s.f.file-sync-client-core] File sync failure: Unable to get
latest-commits from server (https://primary.example.com:8140/file-sync/v1/
latest-commits).
java.net.ConnectException: Connection refused
```

Ignore these errors while the primary is starting. These errors display when Puppet Server restarts as the compilers poll for new code. These errors should stop when the Puppet Server on the primary server finishes restarting.

Set up authentication for Code Manager

To securely deploy environments, Code Manager needs an authentication token for both authentication and authorization.

To generate a token for Code Manager:

1. Assign a user to the deployment role.
2. In the console, create a deployment user.

Tip: Create a dedicated deployment user for Code Manager use.

3. Add the deployment user to the **Code Deployers** role.

Note: This role is automatically created on install, with default permissions for code deployment and token lifetime management.

4. Create a password by clicking **Generate Password**.
5. [Request an authentication token for deployments](#) on page 64

Related information

[Configure puppet-access](#) on page 218

The `puppet-access` command allows users to generate and manage authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server. If you want to use `puppet-access`, ensure it is configured correctly before using it to generate authentication tokens.

Request an authentication token for deployments

Request an authentication token for the deployment user to enable secure deployment of your code.

By default, authentication tokens have a one-hour lifetime. With the `Override default expiry` permission set, you can change the lifetime of the token to a duration better suited for a long-running, automated process.

Generate the authentication token using the `puppet-access` command.

1. From the command line on the primary server, run `puppet-access login --lifetime 180d`. This command both requests the token and sets the token lifetime to 180 days.

Tip: You can add flags to the request specifying additional settings such as the token file's location or the URL for your RBAC API. See [Configuration file settings for puppet-access](#).

2. Enter the username and password of the deployment user when prompted.

The generated token is stored in a file for later use. The default location for storing the token is `~/.puppetlabs/token`. To view the token, run `puppet-access show`.

Test the connection to the control repo.

Related information

[Set a token-specific lifetime](#) on page 222

Tokens have a default lifetime of one hour, but you can set a different lifetime for your token when you generate it. This allows you to keep one token for multiple sessions.

[Generate a token for use by a service](#) on page 222

If you need to generate a token for use by a service and the token doesn't need to be saved, use the `--print` option.

Test the control repo

To make sure that Code Manager can connect to the control repo, test the connection to the repository.

From the command line, run `puppet-code deploy --dry-run`.

- If the control repo is set up properly, this command fetches and displays the number of environments in the control repo.
- If an environment is not set up properly or causes an error, it does not appear in the returned list. Check the Puppet Server log for details about the errors.

Test Code Manager

Test Code Manager by deploying a single test environment.

Deploy a single test environment to test Code Manager:

From the command line, deploy one environment by running `puppet-code deploy my_test_environment --wait`

This deploys the test environment, and then returns deployment results with the SHA (a checksum for the content stored) for the control repository commit.

If the environment deploys and returns the deployment results, Code Manager is correctly configured.

If the deployment does not work, review the configuration steps, or refer to [3e420dfcddc3e961febb280b6440c942184d0bc.ditamap](#) for help.

After Code Manager is fully enabled and configured, you can trigger Code Manager to deploy your environments.

There are several ways to trigger deployments, depending on your needs.

- [Triggering Code Manager on the command line](#) on page 639
- [Triggering Code Manager with a webhook](#) on page 645
- [Triggering Code Manager with custom scripts](#) on page 646

Code Manager settings

After Code Manager is configured, you can adjust its settings in the **PE Master** node group, in the `puppet_enterprise::profile::master` class.

These options are required for Code Manager to work, unless otherwise noted.

puppet_enterprise::profile::master::code_manager_auto_configure

Specifies whether to autoconfigure Code Manager and file sync.

Default: `false`

puppet_enterprise::profile::master::r10k_remote

The location, as a valid URL, for your Git control repository.

Example: `'git@<YOUR.GIT.SERVER.COM>:puppet/control.git'`

puppet_enterprise::profile::master::r10k_private_key

The path to the file containing the private key used to access all Git repositories. Required when using the SSH protocol; optional in all other cases.

Example: `'/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa'`

puppet_enterprise::profile::master::r10k_proxy

Optional proxy used by r10k when accessing the Forge. If empty, no proxy settings are used.

Example: `'http://proxy.example.com:3128'`

puppet_enterprise::profile::master::r10k_trace

Configuration option that includes the r10k stacktrace in the error output of failed deployments when the value is `true`.

Default: `false`

puppet_enterprise::profile::master::versioned_deploys

Optional setting that specifies whether code is updated in versioned code directories instead of blocking requests and overwriting the live code directory.

Default: `false`

Note: For more information, see [Deploying Code without blocking requests to Puppet Server](#) on page 630

puppet_enterprise::master::environment_timeout

Specifies whether and how long environments are cached, which can significantly reduce CPU usage of your Puppet Server. You can specify any of these values:

- 0 – no caching
- unlimited – all environments are cached forever
- a length of time for environments to be cached, for example 30m

Default when Code Manager is enabled: unlimited

Default when Code Manager is *not* enabled: 0

puppet_enterprise::master::environment_timeout_mode

Indicates whether a length of time specified by the `environment_timeout` begins when the environment is created (`from_created`) or when it's last used (`from_last_used`).

Default: `from_created`

Note: For optimal balance of performance and memory use, we recommend setting `environment_timeout_mode` to `from_last_used` and `environment_timeout` to 30m.

To further customize your Code Manager configuration with Hiera, see [Customize Code Manager configuration in Hiera](#) on page 631.

Deploying Code without blocking requests to Puppet Server

When compiling code, Puppet Server typically blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. You can enable *lockless deploys* so the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.

With lockless deploys enabled, each new deploy writes code to versioned directories at `/opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs/puppet-code/`. The standard code directory, `/etc/puppetlabs/code` is reconfigured to `/etc/puppetlabs/puppetserver/code`, which points via symlink to the most recent versioned code directory. If you disable lockless deploys after enabling it, your code directory moves back to the default location.

To conserve disk space, code written to version directories is optimized to reduce duplication, and directories older than the latest and its predecessor are cleaned up after 30 minutes. If you deploy code very frequently, you might prefer to decrease the `versioned-dirs-ttl` setting, which is specified, in minutes, in `file-sync.conf` within each file sync client.

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

Deploy code without blocking requests to Puppet Server

When compiling code, Puppet Server typically blocks requests, including catalog compilation, until file sync is done updating the Puppet code directory. You can enable *lockless deploys* so the file sync client updates code into versioned code directories instead of blocking requests and overwriting the live code directory.

With lockless deploys enabled, each new deploy writes code to versioned directories at `/opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs/puppet-code/`. The standard code directory, `/etc/puppetlabs/code` is reconfigured to `/etc/puppetlabs/puppetserver/code`, which points via symlink to the most recent versioned code directory. If you disable lockless deploys after enabling it, your code directory is moved back to the default location.

To conserve disk space, code written to version directories is optimized to reduce duplication, and directories older than the latest and its predecessor are cleaned up after 30 minutes. If you deploy code very frequently, you might prefer to decrease the `versioned-dirs-ttl` setting, which is specified, in minutes, in `file-sync.conf` within each repository.

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

1. In the console, click **Node groups** and select the **PE Master** node group.
2. On the **Classes** tab, in the `puppet_enterprise::profile::master` class, set `versioned_deploys = true` and click **Add parameter**.
3. Commit changes.
4. Run Puppet on your primary server and all compilers: `puppet agent -t`

The next time you deploy code, your code directory is reconfigured from `/etc/puppetlabs/code` to `/etc/puppetlabs/puppetserver/code`, and versioned code directories are added at `/opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs/puppet-code/`.

System requirements for lockless deploys

Enabling *lockless deploys* increases the disk storage required on your primary server and compilers, because code is written to multiple versioned directories, instead of a single live code directory. Follow these guidelines for estimating your required system capacity.

You can roughly estimate your required disk storage with this equation:

(Size of typical environment)(Number of active environments)

For example, if your typical environment, when deployed, is 200 MB on disk, and you have 25 active environments, your disk storage calculation is $200 \text{ MB} \times 25 = 5,000 \text{ MB}$ or 5 GB.

The number of times you deploy a given environment each day also impacts your disk use. Deploying multiple versions of the same environment uses approximately 25 percent more disk space than deploying multiple unique environments. To estimate the *additional* disk storage required for deploying environments multiple times a day, use this equation:

(Size of typical environment x .25)(Number of environments deployed multiple times per day)(Number of deployments per day)

Expanding on the previous example, if 10 of your active environments are deployed up to 10 times per day, your disk storage calculation is $50 \text{ MB} \times 10 \times 10 = 5,000 \text{ MB}$ or an additional 5 GB of disk space. In total then, you need 10 GB available for your primary server and each compiler.

Note: If you're using the Continuous Delivery for PE impact analysis tool, you might need additional disk space beyond these estimates in order to accommodate the short-lived environments created during impact analysis.

Customize Code Manager configuration in Hieradata

To customize your Code Manager configuration, set parameters with Hieradata.

Add Code Manager parameters in the `puppet_enterprise::master::code_manager` class with Hieradata.

Important:

Do not edit the Code Manager configuration file manually. Puppet manages this configuration file automatically and undoes any manual changes you make.

Related information

[Configure settings with Hieradata](#) on page 161

Hieradata is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hieradata data, Hieradata searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hieradata to manage your PE configuration settings.

Add Code Manager parameters in Hiera

To customize your Code Manager configuration, add parameters to your control repository hierarchy in the `data/common.yaml` file.

1. Add the parameter to `data/common.yaml` in the format:

```
puppet_enterprise::master::code_manager::<parameter>
```

```
puppet_enterprise::master::code_manager::deploy_pool_size: 4
puppet_enterprise::master::code_manager::timeouts_deploy: 300
```

The first parameter in this example increases the size of the default worker pool, and the second reduces the maximum time allowed for deploying a single environment.

2. Run Puppet on the primary server to apply changes.

Configuring post-environment hooks

Configure post-environment hooks to trigger custom actions after your environment deployment.

To configure list of hooks to run after an environment has been deployed, specify the Code Manager `post_environment_hook` setting in Hiera.

This parameter accepts an array of hashes, with the following keys:

- `url`
- `use-client-ssl`

For example, to enable Code Manager to update classes in the console after deploying code to your environments.

```
puppet_enterprise::master::code_manager::post_environment_hooks:
  - url: 'https://console.yourorg.com:4433/classifier-api/v1/update-classes'

  use-client-ssl: true
```

url

This setting specifies an HTTP URL to send a request to, with the result of the environment deploy. The URL receives a POST with a JSON body with the structure:

```
{
  "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
  "file-sync": {
    "environment-commit": "6939889b679fdb1449545c44f26aa06174d25c21",
    "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1"
  },
  "environment": "production",
  "id": 3,
  "status": "complete"
}
```

use-client-ssl

Specifies whether the client SSL configuration is used for HTTPS connections. If the hook destination is a server using the Puppet CA, set this to `true`; otherwise, set it to `false`. Defaults to `false`.

Configuring garbage collection

By default, Code Manager retains environment deployments in memory for one hour, but you can adjust this by configuring garbage collection.

To configure the frequency of Code Manager garbage collection, specify the `deploy_ttl` parameter in Hiera. By default, deployments are kept for one hour.

Specify the time as a string using any of the following suffixes:

- d - days
- h - hours
- m - minutes
- s - seconds
- ms - milliseconds

For example, a value of 30d would configure Code Manager to keep the deployment in memory for 30 days, and a value of 48h would maintain the deployment in memory for 48 hours.

If the value of `deploy-ttl` is less than the combined values of `timeouts_fetch`, `timeouts_sync`, and `timeouts_deploy`, all completed deployments are retained indefinitely, which might significantly slow the performance of Code Manager over time.

Configuring Forge settings

To configure how Code Manager downloads modules from the Forge, specify `forge_settings` in Hiera.

This parameter configures where Forge modules should be installed from, and sets a proxy for all Forge interactions. The `forge_settings` parameter accepts a hash with the following values:

- `baseurl`
- `proxy`

baseurl

Indicates where Forge modules should be installed from. Defaults to `https://forgeapi.puppetlabs.com`.

```
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: 'https://private-forge.mysite'
```

proxy

Sets the proxy for all Forge interactions.

This setting overrides the global `proxy` setting on Forge operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. See the global `proxy` setting for more information and examples.

```
proxy: 'http://proxy.example.com:3128'
```

Configuring Git settings

To configure Code Manager to use a private key, a proxy, or multiple repositories with Git, specify the `git_settings` parameter.

The `git_settings` parameter accepts a hash of the following settings:

- `private-key`
- `proxy`
- `repositories`

Note: You cannot use the `git_settings` hash with the default Code Manager settings for `r10k_private_key`. To avoid errors, remove the `r10k_private_key` from the `puppet_enterprise::profile::master` class.

private-key

Specifies the file containing the default private key used to access control repositories, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`. This file must have read permissions for the `pe-puppet` user. The SSH key cannot require a password. This setting is required, but by default, the value is supplied in the `puppet_enterprise::profile::master` class.

proxy

Sets a proxy specifically for Git operations that use an HTTP(S) transport.

This setting overrides the global `proxy` setting on Git operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. To set a proxy for a specific Git repository only, set `proxy` in the `repositories` subsetting of `git_settings`. See the global `proxy` setting for more information and examples.

```
proxy: 'http://proxy.example.com:3128'
```

repositories

Specifies a list of repositories and their respective private keys. Use this setting if you want to use multiple control repos.

To use multiple control repos, the `sources` setting and the `repositories` setting must match. Accepts the following settings:

- `remote`: The repository to which these settings apply.
- `private-key`: The file containing the private key to use for this repository. This file must have read permissions for the `pe-puppet` user.
- `proxy`: The proxy setting allows you to set or override the global proxy setting for a single, specific repository. See the global proxy setting for more information and examples.

The `repositories` setting accepts a hash in the following format:

```
repositories:
  - remote: "jfkennedy@gitserver.puppetlabs.net:repositories/repo_one.git"
    private-key: "/test_keys/jfkennedy"
  - remote: "whaft@gitserver.puppetlabs.net:repositories/repo_two.git"
    private-key: "/test_keys/whaft"
  - remote: "https://git.example.com/git_repos/environments.git"
    proxy: "https://proxy.example.com:3128"
```

Configuring proxies

To configure proxy servers, use the `proxy` setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

proxy

To set a proxy for all operations occurring over an HTTP(S) transport, set the global `proxy` setting. You can also set an authenticated proxy with either Basic or Digest authentication.

To override this setting for Git or Forge operations only, set the `proxy` subsetting under the `git_settings` or `forge_settings` parameters. To override for a specific Git repository, set a `proxy` in the `repositories` list of `git_settings`. To override this setting with no proxy for Git, Forge, or a particular repository, set that specific `proxy` setting to an empty string.

In this example, the first proxy is set up without authentication, and the second proxy uses authentication:

```
proxy: 'http://proxy.example.com:3128'
proxy: 'http://user:password@proxy.example.com:3128'
```

Configuring sources

If you are managing more than one repository with Code Manager, specify a map of your source repositories.

Use the `source` parameter to specify a map of sources. Configure this setting if you are managing more than just Puppet environments, such as when you are also managing Hiera data in its own control repository.

To use multiple control repos, the `sources` setting and the `repositories` setting must match.

If `sources` is set, you cannot use the global `remote` setting. If you are using multiple sources, use the `prefix` option to prevent collisions.

The `sources` setting accepts a hash with the following subsettings:

- `remote`
- `prefix`

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: "testing"
```

remote

Specifies the location from which you fetch the source repository. The remote must be able to be fetched without any interactive input. That is, you cannot be prompted for usernames or passwords in order to fetch the remote.

Accepts a string that is any valid URL that `r10k` can clone, such as `git://git-server.site/my-org/main-modules`.

prefix

Specifies a string to prefix to environment names. Alternatively, if `prefix` is set to `true`, the source's name is used. This prevents collisions when multiple sources are deployed into the same directory.

For example, with two "main-modules" environments set up in the same base directory, the `prefix` setting differentiates the environments: the first is named "myorg-main-modules", and the second is "testing-main-modules".

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: "testing"
```

Code Manager parameters

Code Manager parameters

Parameter	Description	Value	Default
<code>remote</code>	The location of the Git control repository. Either the <code>remote</code> or <code>sources</code> parameters must be specified. If <code>remote</code> is already specified in the <code>puppet_enterprise::profile::master</code> class, that value is used here. If both the <code>sources</code> and <code>remote</code> keys are specified, the <code>sources</code> key overrides <code>remote</code> .	A string that is a valid SSH URL for your Git remote.	No default.
<code>authenticate_webhook</code>	Turns on RBAC authentication for the <code>/v1/webhook</code> endpoint.	Boolean.	<code>true</code>

Parameter	Description	Value	Default
cachedir	The path to the location where Code Manager caches Git repositories.	A valid file path.	/opt/puppetlabs/server/data/code-manager/cache.
certname	The certname of the Puppet signed certs to use for SSL.	A valid certname.	Defaults to \$::clientcert.
data	The path to the directory where Code Manager stores internal file content.	A valid file path.	/opt/puppetlabs/server/data/code-manager
deploy_pool_size	Specifies the number of threads in the worker pool; this dictates how many deploy processes can run in parallel.	An integer.	2
download_pool_size	Specifies the number of threads used to download modules.	An integer.	4
deploy_ttl	Specifies the length of time completed deployments are retained before garbage collection. For usage details, see configuring garbage collection.	The time as a string, using any of the following suffixes: <ul style="list-style-type: none"> • d - days • h - hours • m - minutes • s - seconds • ms - milliseconds 	1h (one hour)
hostcrl	The path to the SSL CRL.	A valid file path.	\$puppet_enterprise::params::h
localcacert	The path to the SSL CA cert.	A valid file path.	\$puppet_enterprise::params::l

Parameter	Description	Value	Default
<code>post_environment_hooks</code>	<p>A list of hooks to run after an environment has been deployed. Specifies:</p> <ul style="list-style-type: none"> • an HTTP URL to send deployment results to, • Whether to use client SSL for HTTPS connections <p>For usage details, see configuring post-environment hooks.</p>	<p>An array of hashes that can include the keys:</p> <ul style="list-style-type: none"> • <code>url</code> • <code>use-client-ssl</code> 	No default.
<code>timeouts_deploy</code>	Maximum execution time (in seconds) for deploying a single environment.	An integer	600
<code>timeouts_fetch</code>	Maximum execution time (in seconds) for updating the control repo state.	An integer.	30
<code>timeouts_hook</code>	Maximum time (in seconds) to wait for a single post-environment hook URL to respond. Used for both the socket connect timeout and the read timeout, so the longest total timeout would be twice the specified value.	An integer.	30
<code>timeouts_shutdown</code>	Maximum time (in seconds) to wait for in-progress deploys to complete when shutting down the service.	An integer.	610
<code>timeouts_wait</code>	Maximum time that a request sent with a <code>wait</code> parameter should wait for each environment before timing out.	An integer.	700
<code>timeouts_sync</code>	Maximum time (in seconds) that a request sent with a <code>wait</code> parameter must wait for all compilers to receive deployed code before timing out.	An integer.	300

Parameter	Description	Value	Default
webserver_ssl_host	The host that Code Manager listens on.	An IP address	0.0.0.0
webserver_ssl_port	The port that Code Manager listens on. Port 8170 must be open if you're using Code Manager.	A port number.	8170

r10k specific parameters

These parameters are specific to r10k, which Code Manager uses in the background.

Parameter	Description	Value	Default
environmentdir	The single directory to which Code Manager deploys all sources. If <code>file_sync_auto_commit</code> is set to <code>true</code> , this defaults to <code>/etc/puppetlabs/code-staging/environments</code> . See <code>file_sync_auto_commit</code> .	Directory	<code>/etc/puppetlabs/code-staging/environments</code>
forge_settings	Contains settings for downloading modules from the Forge. See Configuring Forge settings for usage details.	Accepts a hash of: <ul style="list-style-type: none"> <code>baseurl</code> <code>proxy</code> 	No default.
invalid_branches	Specifies, for all sources, how branch names that cannot be cleanly mapped to Puppet environments are handled.	<ul style="list-style-type: none"> <code>'correct'</code>: Replaces non-word characters with underscores and gives no warning. <code>'error'</code>: Ignores branches with non-word characters and issues an error. 	<code>'error'</code>

Parameter	Description	Value	Default
<code>git_settings</code>	<p>Configures settings for Git:</p> <ul style="list-style-type: none"> Specifies the file containing the default private key used to access control repositories. Sets or overrides the global proxy setting specifically for Git operations that use an HTTP(S) transport. Specifies a list of repositories and their respective private keys. <p>See Configuring Git settings for usage details.</p>	<p>Accepts a hash of:</p> <ul style="list-style-type: none"> <code>private-key</code> <code>proxy</code> <code>repositories</code> 	Default private-key value as set in console. No other default settings.
<code>proxy</code>	<p>Configures a proxy server to use for all operations that occur over an HTTP(S) transport.</p> <p>See Configuring proxies for usage details.</p>	<p>Accepts:</p> <ul style="list-style-type: none"> A proxy server. An authenticated proxy with Basic or Digest authentication. An empty value. 	No default.
<code>sources</code>	<p>Specifies a map of sources to be passed to <code>r10k</code>. Use if you are managing more than just Puppet environments.</p> <p>See Configuring sources for usage details.</p>	<p>A hash of:</p> <ul style="list-style-type: none"> <code>remote</code> <code>prefix</code> 	No default.

Triggering Code Manager on the command line

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

Installing and configuring puppet-code

PE automatically installs and configures the `puppet-code` command on your primary server as part of the included PE client tools package. You can also set up `puppet-code` on an agent node or on a workstation, customize configuration for different users, or change the configuration settings.

The global configuration settings for Linux and macOS systems are in a JSON file at `/etc/puppetlabs/client-tools/puppet-code.conf`. The default configuration file looks something like:

```
{
  "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
  "token-file": "~/.puppetlabs/token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

Important:

On a PE-managed machine, Puppet manages this file for you. Do not manually edit this file, because Puppet overwrites your new values the next time it runs.

Additionally, you can set up `puppet-code` on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files. For instructions, see the related topic about advanced configuration of `puppet-code`.

You can also override existing configuration settings on a case-by-case basis on the command line. When you deploy environments with the `puppet-code deploy` command, you can specify either an alternative config file or particular config settings directly in the command. For examples, see the related topic about deploying environments with `puppet code`.

Windows paths

The global `puppet-code` configuration file on Windows systems is located at `C:\ProgramData\PuppetLabs\client-tools\puppet-code.conf`.

The default configuration file looks something like:

```
{
  "cacert": "C:\\ProgramData\\PuppetLabs\\puppet\\etc\\ssl\\certs\\ca.pem",
  "token-file": "C:\\Users\\<username>\\.puppetlabs\\token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

Configuration precedence and puppet-code

There are several ways to configure `puppet-code`, but some configuration methods take precedence over others.

If no other configuration is specified, `puppet-code` uses the settings in the global configuration file. User-specific configuration files override the global configuration file.

If you specify a configuration file on the command line, Puppet temporarily uses that configuration file **only** and does not read the global or user-specific config files at all.

Finally, if you specify individual configuration options directly on the command line, those options temporarily take precedence over *any* configuration file settings.

Deploying environments with puppet-code

To deploy environments with the `puppet-code` command, use the `deploy` action, either with the name of a single environment or with the `--all` flag.

The `deploy` action deploys the environments, but returns only deployment *queuing* results by default. To view the results of the deployment itself, add the `--wait` flag.

The `--wait` flag deploys the specified environments, waits for file sync to complete code deployment to the live code directory and all compilers, and then returns results. In deployments that are geographically dispersed or have a large quantity of environments, completing code deployment can take up to several minutes.

The resulting message includes the deployment signature, which is the commit SHA of the control repo used to deploy the environment. The output also includes two other SHAs that indicate that file sync is aware that the environment has been newly deployed to the code staging directory.

To temporarily override default, global, and user-specific configuration settings, specify the following configuration options on the command line:

- `--cacert`
- `--token-file, -t`
- `--service-url`

Alternately, you can specify a custom `puppet-code.conf` configuration file by using the `--config-file` option.

Running puppet-code on Windows

If you're running these commands from a managed or non-managed Windows workstation, you must specify the full path to the command.

```
C:\Program Files\Puppet Labs\Client\bin\puppet code deploy mytestenvironment
--wait
```

Deploy environments on the command line

To deploy environments, use the `puppet-code deploy` command, specifying the environments to deploy.

To deploy environments, on the command line, run `puppet-code deploy`, specifying the environment.

Specify the environment by name. To deploy all environments, use the `--all` flag.

Optionally, you can specify the `--wait` flag to return results after the deployment is finished. Without the `--wait` flag, the command returns only queuing results.

```
puppet-code deploy myenvironment --wait
```

```
puppet-code deploy --all --wait
```

Both of these commands deploy the specified environments, and then return deployment results with a control repo commit SHA for each environment.

Related information

[Reference: puppet-code command](#) on page 643

The `puppet-code` command accepts options, actions, and `deploy` action options.

Deploy with a custom configuration file

You can deploy environments with a custom configuration file that you specify on the command line.

To deploy all environments using the configuration settings in a specified config file, run the command `puppet-code deploy` command with a `--config-file` flag specifying the location of the config file.

```
puppet-code --config-file ~/.puppetlabs/myconfigfile/puppet code.conf deploy
--all
```

Deploy with command-line configuration settings

You can override an existing configuration setting on a per-use basis by specifying that setting on the command line.

Specify the setting, which is used on this deployment only, on the command line.

```
puppet-code --service-url "https://puppet.example.com:8170/code-manager"
deploy mytestenvironment
```

This example deploys 'mytestenvironment' using global or user-specific config settings (if set), except for `--service-url`, for which it uses the value specified on the command line ("https://puppet.example.com:8170/code-manager").

Advanced puppet-code configuration

You can set up the `puppet-code` command on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files.

You can set up the `puppet-code` command on an agent node or on a workstation not managed by PE. And on any machine, you can set up user-specific config files.

The `puppet-code.conf` file is a JSON configuration file for the `puppet-code` command. For Linux or Mac OS X operating systems, it looks something like:

```
{
  "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
  "token-file": "~/.puppetlabs/token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

For Windows systems, use the entire Windows path, such as:

```
{
  "cacert": "C:\\ProgramData\\PuppetLabs\\puppet\\etc\\ssl\\certs\\ca.pem",
  "token-file": "C:\\Users\\<username>\\.puppetlabs\\token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

Related information

[Installing PE client tools](#) on page 138

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

Configure puppet-code on agents and workstations

To use `puppet-code` on an agent node or on a workstation that is not managed by PE, install the client tools package and configure `puppet-code` on that machine.

Before you begin

Download and install the client tools package. See the client tools documentation for instructions.

Create a config file called `puppet-code.conf` in the client tools directory.

- For Linux and Mac OS X systems, the default client tools directory is `/etc/puppetlabs/client-tools`
- For Windows systems, the default client tools directory is `C:\ProgramData\PuppetLabs\client-tools`

Configure puppet-code for different users

On any machine, whether it is a primary server, an agent, or a workstation not managed by PE, you can set up specific `puppet-code` configurations for specific users.

Before you begin

If PE is **not** installed on the workstation you are configuring, see instructions for configuring `puppet-code` on agents and workstations.

1. Create a `puppet-code.conf` file in the user's client tools directory.
 - For Linux or Mac OS X systems, place the file in the user's `~/.puppetlabs/client-tools/`
 - For Windows systems, place the file in the default user config file location: `C:\Users\<username>\.puppetlabs\ssl\certs\ca.pem`
2. In the user's `puppet-code.conf` file, specify the `cacert`, `token-file`, and `service-url` settings in JSON format.

Reference: puppet-code command

The `puppet-code` command accepts options, actions, and deploy action options.

Usage: `puppet-code` [global options] <action> [action options]

Global puppet-code options

Global options set the behavior of the `puppet-code` command on the command line.

Option	Description	Accepted arguments
<code>--help, -h</code>	Prints usage information for <code>puppet-code</code> .	none
<code>--version, -V</code>	Prints the application version.	none
<code>--log-level, -l</code>	Sets the log verbosity. It accepts one log level as an argument.	log levels: none, trace, debug, info, warn, error, fatal.
<code>--config-file, -c</code>	Sets a <code>puppet-code.conf</code> file that takes precedence over all other existing <code>puppet-code.conf</code> files.	A path to a <code>puppet-code.conf</code> file.
<code>--cacert</code>	Sets a Puppet CA certificate that overrides the <code>cacert</code> setting in any configuration files.	A path to the location of a CA Certificate.
<code>--token-file, -t</code>	Sets an authentication token that overrides the <code>token-file</code> setting in any configuration files.	A path to the location of the authentication token.
<code>--service-url</code>	Sets a base URL for the Code Manager service, overriding <code>service-url</code> settings in any configuration files.	A valid URL to the service.

puppet-code actions

The `puppet-code` command can perform print, deploy, and status actions.

Action	Result	Arguments	Options
<code>puppet-code print-config</code>	Prints out the resolved <code>puppet-code</code> configuration.	none	none
<code>puppet-code deploy</code>	Runs remote code deployments with the Code Manager service. By default, returns only deployment queuing results.	Accepts either the name of a single environment or the <code>--all</code> flag.	<code>--wait, -w</code> ; <code>--all</code> ; <code>--dry-run</code> ; <code>--format, -F</code>
<code>puppet-code status</code>	Checks whether Code Manager and file sync are responding. By default, details are returned at the info level.	Accepts log levels none, trace, info, warn, error, fatal.	none

puppet-code deploy action options

Modify the `puppet-code deploy` action with action options.

Option	Description
<code>--wait, -w</code>	Causes <code>puppet-code deploy</code> to: <ol style="list-style-type: none"> 1. Start a deployment, 2. Wait for the deployment to complete, 3. Wait for file sync to deploy the code to all compilers, and 4. Return the deployment signature with control repo commit SHAs for each environment. <p>The return output also includes two other SHAs that indicate that file sync is aware that the environment has been newly deployed to the code-staging directory.</p>
<code>--all</code>	Tells <code>puppet-code deploy</code> to start deployments for all Code Manager environments.
<code>--dry-run</code>	Tests the connections to each configured remote and, if successfully connected, returns a consolidated list of the environments from all remotes. The <code>--dry-run</code> flag implies both <code>--all</code> and <code>--wait</code> .

puppet-code.conf configuration settings

Temporarily specify `puppet-code.conf` configuration settings on the command line.

Setting	Controls	Linux and Mac OS X default	Windows default
<code>cacert</code>	Specifies the path to the Puppet CA certificate to use when connecting to the Code Manager service over SSL.	<code>/etc/puppetlabs/puppet/ssl/certs/ca.pem</code>	<code>C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem</code>
<code>token-file</code>	Specifies the location of the file containing the authentication token for Code Manager.	<code>~/.puppetlabs/token</code>	<code>C:\Users\<username>\.puppetlabs\token</code>
<code>service-url</code>	Specifies the base URL of the Code Manager service	<code>https://PRIMARY_HOSTNAME:8170/code-manager</code>	<code>https://PRIMARY_HOSTNAME:8170/code-manager</code>

Triggering Code Manager with a webhook

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. The webhook is the simplest way to trigger Code Manager.

Custom scripts are a good alternative if you have requirements such as existing continuous integration systems (including Continuous Delivery for Puppet Enterprise (PE)), privately hosted Git repos, or custom notifications. For information about writing a script to trigger Code Manager, see the related topic about creating custom scripts.

Code Manager supports webhooks for GitHub, Bitbucket Server (formerly Stash), Bitbucket, and Team Foundation Server. The webhook must only be used by the control repository. It can't be used by any other repository (for example, other internal component module repositories).

Important: Code Manager webhooks are not compatible with Continuous Delivery for PE. If your organization uses Continuous Delivery for PE, you must use a method other than webhooks to deploy environments.

Creating a Code Manager webhook

To set up the webhook to trigger environment deployments, you must create a custom URL, and then set up the webhook with your Git host.

Creating a custom URL for the Code Manager webhook

To trigger deployments with a webhook, you'll need a custom URL to enable communication between your Git host and Code Manager.

Code Manager supports webhooks for GitHub, Bitbucket Server (formerly Stash), Bitbucket, GitLab (Push events only), and Team Foundation Server (TFS). To use the GitHub webhook with the Puppet signed cert, disable SSL verification.

To create the custom URL for your webhook, use the following elements:

- The name of the primary server (for example, `code-manager.example.com`).
- The Code Manager port (for example, 8170).
- The endpoint (`/code-manager/v1/webhook/`).
- Any relevant query parameters (for example, `type=github`).
- Your authentication token for deployments (`token=<TOKEN>`), passed with the `token` query parameter. To generate a token, see [Request an authentication token for deployments](#) on page 64.

For example, the URL for a GitHub webhook might look like this:

```
https://code-manager.example.com:8170/code-manager/v1/webhook?
type=github&token=<TOKEN>
```

The URL for a version 5.4 or later Bitbucket Server webhook might look something like this:

```
https://code-manager.example.com:8170/code-manager/v1/webhook?
type=bitbucket-server&prefix=dev&token=<TOKEN>
```

With the complete token attached, a GitHub URL looks something like this:

```
https://code-manager.example.com:8170/code-manager/v1/webhook?
type=github&token=0WJ4YPJVyQz26xm3X2I10ihb7MUa6812CZWjxM3vt4mQ
```

Code Manager webhook query parameters

The following query parameters are permitted in the Code Manager webhook.

The `token` query is mandatory, unless you disable `authenticate_webhook` in the Code Manager configuration.

- `type`: Required. Specifies which type of post body to expect. Accepts:
 - `github`: GitHub
 - `gitlab`: GitLab
 - `bitbucket-server`: Bitbucket Server version 5.4 or later (formerly Stash)
 - `bitbucket`: Bitbucket
 - `tfs-git`: Team Foundation Server (resource version 1.0 is supported)
- `prefix`: Specifies a prefix for converting branch names to environments.
- `token`: Specifies the entire PE authorization token.

Setting up the Code Manager webhook on your Git host

Enter the custom URL you created for Code Manager into your Git server's webhook form as the payload URL.

The content type for webhooks is JSON.

Exactly how you set up your webhook varies, depending on where your Git repos are hosted. For example, in your GitHub repo, click on **Settings > Webhooks & services** to enter the payload URL and enter `application/json` as the content type.

Tip: On Bitbucket Server, the server configuration menu has settings for both "hooks" and "webhooks." To set up Code Manager, use the webhooks configuration. For proper webhook function with Bitbucket Server, make sure you are using the Bitbucket Server 5.4 or later and the latest fix version of PE.

After you've set up your webhook, your Code Manager setup is complete. When you commit new code and push it to your control repo, the webhook triggers Code Manager, and your code is deployed.

Testing and troubleshooting a Code Manager webhook

To test and troubleshoot your webhook, review your Git host logs or check the Code Manager troubleshooting guide.

Each of the major repository hosting services (such as GitHub or Bitbucket) provides a way to review the logs for your webhook runs, so check their documentation for instructions.

For other issues, check the Code Manager troubleshooting for some common problems and troubleshooting tips.

Triggering Code Manager with custom scripts

Custom scripts are a good way to trigger deployments if you have requirements such as existing continuous integration systems, privately hosted Git repos, or custom notifications.

Alternatively, a webhook provides a simpler way to trigger deployments automatically.

To create a script that triggers Code Manager to deploy your environments, you can use either the `puppet-code` command or a `curl` statement that hits the endpoints. We recommend the `puppet-code` command.

Either way, after you have composed your script, you can trigger deployment of new code into your environments. Commit new code, push it to your control repo, and run your script to trigger Code Manager to deploy your code.

All of these instructions assume that you have enabled and configured Code Manager.

Related information

[Request an authentication token for deployments](#) on page 64

Request an authentication token for the deployment user to enable secure deployment of your code.

[Code Manager API](#) on page 651

Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

Deploying environments with the `puppet-code` command

The `puppet-code` command allows you to trigger environment deployments from the command line. You can use this command in your custom scripts.

For example, to deploy all environments and then return deployment results with commit SHAs for each of your environments, incorporate this command into your script:

```
puppet-code deploy --all --wait
```

Related information

[Triggering Code Manager on the command line](#) on page 639

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

Deploying environments with a `curl` command

To trigger Code Manager code deployments with a custom script, compose a `curl` command to hit the Code Manager/`deploys` endpoint.

To deploy environments with a `curl` command in your custom script, compose a `curl` command to hit the `/v1/deploys` endpoint.

Specify either the `"deploy-all"` key, to deploy all configured environments, or the `"environments"` key, to deploy a specified list of environments. By default, Code Manager returns queuing results immediately after accepting the deployment request.

If you prefer to get complete deployment results after Code Manager finishes processing the deployments, specify the optional `"wait"` key with your request. In deployments that are geographically dispersed or have a large number of environments, completing code deployment can take up to several minutes.

For complete information about Code Manager endpoints, request formats, and responses, see the Code Manager API documentation.

You must include a custom URL for Code Manager and a PE authentication token in any request to a Code Manager endpoint.

Creating a custom URL for Code Manager scripts

To trigger deployments with a `curl` command in a custom script, you'll need a custom Code Manager URL. This URL is composed of:

- Your primary server name (for example, `primary.example.com`)
- The Code Manager port (for example, `8170`)
- The endpoint you want to hit (for example, `/code-manager/v1/deploys/`)
- The authentication token you generated earlier (`token=<TOKEN>`)

A typical URL for use with a `curl` command might look something like this:

```
https://primary.example.com:8170/code-manager/v1/deploys&token=<TOKEN>
```

Attaching an authentication token

You must attach a PE authentication token to any request to Code Manager endpoints. You can either:

- Specify the path to the token with `puppet-access show` in the body of your request. For example:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
```

```
data='{ "environments": [ "production" ], "wait": true }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

- Attach the entire token to your Code Manager URL using the `token` query parameter. For example:

```
https://$(puppet config print server):8170/code-manager/v1/webhook?
type=github&token=<TOKEN>
```

If you are using a curl command to hit the `/webhook` endpoint directly, you must attach the entire token.

Checking deployment status with a curl command

You can check the status of a deployment by hitting the `status` endpoint.

To use a curl command in your custom script, compose a curl command to hit the status endpoint. You must incorporate a custom URL for Code Manager in the script.

Troubleshooting Code Manager

Code Manager requires coordination between multiple components, including `r10k` and the file sync service. If you have issues with Code Manager, check that these components are functioning.

Code Manager logs

Code Manager logs to the Puppet Server log. By default, this log is in `/var/log/puppetlabs/puppetserver/puppetserver.log`. For more information about working with the logs, see the [Puppet Server logs](#) documentation.

Check Code Manager status

Check the status of Code Manager and file sync if your deployments are not working as expected, or if you need to verify that Code Manager is enabled before running a dependent command.

The command `puppet-code status` verifies that Code Manager and file sync are responding. The command returns the same information as the Code Manager status endpoint. By default, the command returns details at the `info` level; `critical` and `debug` aren't supported.

The following table shows errors that might appear in the `puppet-code status` output.

Error	Cause
Code Manager couldn't connect to the server	<code>pe-puppetserver</code> process isn't running
Code Manager reports invalid configuration	Invalid configuration at <code>/etc/puppetlabs/puppetserver/conf.d/code-manager.conf</code>
File sync storage service reports unknown status	Status callback timed out

Test the connection to the control repository

The control repository controls the creation of environments, and ensures that the correct versions of all the necessary modules are installed. The primary server must be able to access and clone the control repo as the `pe-puppet` user.

To make sure that Code Manager can connect to the control repo, run:

```
puppet-code deploy --dry-run
```

If the connection is set up correctly, this command returns a list of all environments in the control repo or repos. If the command completes successfully, the SSH key has the correct permissions, the Git URL for the repository is correct, and the `pe-puppet` user can perform the operations involved.

If the connection is not working as expected, Code Manager reports an `Unable to determine current branches for Git source` error.

The unsuccessful command also returns a path on the primary server that you can use for debugging the SSH key and Git URL.

Check the Puppetfile for errors

Check the Puppetfile for syntax errors and verify that every module in the Puppetfile can be installed from the listed source. To do this, you need a copy of the Puppetfile in a temporary directory.

Create a temporary directory `/var/tmp/test-puppetfile` on the primary server for testing purposes, and place a copy of the Puppetfile into the temporary directory.

You can then check the syntax and listed sources in your Puppetfile.

Check Puppetfile syntax

To check the Puppetfile syntax, run `r10k puppetfile check` from within the temporary directory.

If you have Puppetfile syntax errors, correct the syntax and test again. When the syntax is correct, the command prints "Syntax OK".

Check the sources listed in the Puppetfile

To test the configuration of all sources listed in your Puppetfile, perform a test installation. This test installs the modules listed in your Puppetfile into a modules directory in the temporary directory.

In the temporary directory, run the following command:

```
sudo -H -u pe-puppet bash -c \
  '/opt/puppetlabs/puppet/bin/r10k puppetfile install'
```

This installs all modules listed in your Puppetfile, verifying that you can access all listed sources. Take note of **all** errors that occur. Issues with individual modules can cause issues for the entire environment. Errors with individual modules (such as Git URL syntax or version issues) show up as general errors for that module.

If you have modules from private Git repositories requiring an SSH key to clone the module, check that you are using the SSH Git URL and not the HTTPS Git URL.

After you've fixed errors, test again and fix any further errors, until all errors are fixed.

Run a deployment test

Manually run a full `r10k` deployment to check not only the Puppetfile syntax and listed host access, but also whether the deployment works through `r10k`.

This command attempts a full `r10k` deployment based on the `r10k.yaml` file that Code Manager uses. This test writes to the code staging directory only. This does not trigger a file sync and is to be used only for ad-hoc testing.

Run this deployment test with the following command:

```
sudo -H -u pe-puppet bash -c \
  '/opt/puppetlabs/puppet/bin/r10k deploy environment -c /opt/puppetlabs/
server/data/code-manager/r10k.yaml -p -v debug'
```

If this command completes successfully, the `/etc/puppetlabs/code-staging` directory is populated with directory-based environments and all of the necessary modules for every environment.

If the command fails, the error is likely in the Code Manager settings specific to `r10k`. The error messages indicate which settings are failing.

Monitor logs for webhook deployment trigger issues

Issues that occur when a Code Manager deployment is triggered by the webhook can be tricky to isolate. Monitor the logs for the deployment trigger to find the issue.

Deployments triggered by a webhook in Stash/Bitbucket, GitLab, or GitHub are governed by authentication and hit each service's `/v1/webhook` endpoint.

If you are using a GitLab version older than 8.5.0, Code Manager webhook authentication does not work because of the length of the authentication token. To use the webhook with GitLab, either disable authentication or update GitLab.

Note: If you disable webhook authentication, it is disabled **only** for the `/v1/webhook` endpoint. Other endpoints (such as `/v1/deploys`) are still controlled by authentication. There is no way to disable authentication on any other Code Manager endpoint.

To troubleshoot webhook issues, follow the Code Manager webhook instructions while monitoring the Puppet Server log for successes and errors. To do this, open a terminal window and run:

```
tail -f /var/log/puppetlabs/puppetserver/puppetserver.log
```

Watch the log closely for errors and information messages when you trigger the deployment. The `puppetserver.log` file is the only location these errors appear.

If you cannot determine the problem with your webhook in this step, manually deploy to the `/v1/deploys` endpoint, as described in the next section.

Monitor logs for /v1/deploys deployment trigger issues

Issues that occur when a Code Manager deployment is triggered by the `/v1/deploys` endpoint can be tricky to isolate. Monitor the logs for the deployment trigger to find the issue.

Before you trigger a deployment to the `/v1/deploys` endpoint, generate an authentication token. Then deploy one or more environments with the following command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{ "environments": ["production"], "wait": true }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

This request waits for the deployment and sync to compilers to complete (`"wait": true`) and so returns errors from the deployment.

Alternatively, you can deploy **all** environments with the following curl command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{ "deploy-all": true, "wait": true }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

Monitor the `console-services.log` file for any errors that arise from this curl command.

```
tail -f /var/log/puppetlabs/console-services/console-services.log
```

Code deployments time out

If your environments are heavily loaded, code deployments can take a long time, and the system can time out before deployment is complete.

If your deployments are timing out too soon, increase your `timeouts_deploy` key. You might also need to increase `timeouts_shutdown`, `timeouts_sync`, and `timeouts_wait`.

File sync stops when Code Manager tries to deploy code

Code Manager code deployment involves accessing many small files. If you store your Puppet code on network-attached storage, slow network or backend hardware can result in poor deployment performance.

Tune the network for many small files, or store Puppet code on local or direct-attached storage.

Classes are missing after deployment

After a successful code deployment, a class you added isn't showing in the console.

If your code deployment works, but a class you added isn't in the console:

1. Check on disk to verify that the environment folder exists.
2. Check your node group in the **Edit node group metadata** box to make sure it's using the correct environment.
3. Refresh classes.
4. Refresh your browser.

Code Manager API

Use Code Manager endpoints to deploy code and query environment deployment status on your primary server and compilers without direct shell access.

Authentication

All Code Manager endpoint requests require token-based authentication. For `/deploys` endpoints, you can pass this token as either an HTTP header or as a query parameter. For the `/webhook` endpoint, you must attach the entire token with the `token` query parameter.

To generate an authentication token, from the command line, run `puppet-access login`. This command stores the token in `~/.puppetlabs/token`. To learn more about authentication for Code Manager use, see the related topic about requesting an authentication token.

Pass this token as either an `X-Authentication` HTTP header or as the `token` query parameter with your request. If you pass the token as a query parameter, it might show up in access logs.

For example, an HTTP header for the `/deploys` endpoint looks something like:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{ "environments": ["production"], "wait": true}'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

```
$pe_headers = @{
    "X-Authentication" = "AMilWlel2Ybd2Lqu-hdhHaLWckfvGsgl8AUgKdwzixwe" ;
}

$pe_uri = "https://puppet.winops2019.automationdemos.com:8170/code-manager/v1/deploys"

$pe_codemgr_hash = [ordered]@{
```

```

        environments = @("production")
    }

# Generate JSON object from pe_codemgr_hash
$JSON = $pe_codemgr_hash | convertto-json -compress
Invoke-RestMethod -SkipCertificateCheck -ContentType "application/json" -uri
    $pe_uri -Method POST -Body $JSON -Headers $pe_headers

```

Passing the token with the token query parameter might look like:

```

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/webhook?
type=github&token=$(puppet-access show)"
data='{ "environments": ["production"], "wait": true }'

curl --insecure --header "$type_header" --header "$auth_header" --request
    POST "$uri" --data "$data"

```

```

# Common Data
$token = "AMilWlel2Ybd2Lqu-hdhHaLWCkfvgSgl8AUgKdwzixwe"
$pe_uri = "https://puppet.winops2019.automationdemos.com:8170/code-manager"

# Example 1 - deploys

$pe_headers = @{
    "X-Authentication" = "$token" ;
}

$pe_codemgr_hash = [ordered]@{
    environments = @("production")
}

# Generate JSON object from pe_codemgr_hash
$JSON = $pe_codemgr_hash | convertto-json -compress
$pe_ex1_uri = "$pe_uri/v1/deploys"
Invoke-RestMethod -SkipCertificateCheck -ContentType "application/json" -uri
    $pe_ex1_uri -Method POST -Body $JSON -Headers $pe_headers

# Example 2 - Web hook

$pe_ex2_uri = "$pe_uri/v1/webhook?token=$token&type=github"
Write-Output "PE Example 2 URI: $pe_ex2_uri"
Invoke-RestMethod -SkipCertificateCheck -uri $pe_ex2_uri -Method POST -
    ContentType "application/json"

```

Related information

[API index](#) on page 26

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

[Request an authentication token for deployments](#) on page 64

Request an authentication token for the deployment user to enable secure deployment of your code.

POST /v1/deploys

Use the Code Manager/deploys endpoint to queue code deployments.

Request format

The body of the POST /deploys request must be a JSON object that includes the authentication token and a URL containing:

- The name of the primary server, such as `primary.example.com`
- The Code Manager port, such as 8170.
- The endpoint.

The request must also include either:

- The "environments" key, with or without the "wait" key.
- The "deploy-all" key, with or without the "wait" key.

Key	Definition	Values
"environments"	Specifies the environments for which to queue code deployments. If not specified, the "deploy-all" key must be specified.	Accepts an array of strings specifying environments.
"deploy-all"	Specifies whether Code Manager deploys all environments or not. If specified <code>true</code> , Code Manager determines the most recent list of environments and queues a deploy for each one. If specified <code>false</code> or not specified, the "environments" key must be specified with a list of environments.	<code>true</code> , <code>false</code>
"deploy-modules"	Specifies whether Code Manager deploys modules from the Puppetfile. If specified <code>true</code> or not specified, Code Manager deploys modules. If specified <code>false</code> , modules aren't deployed. Note: To ensure environments are populated before use, modules are deployed on the first deployment of an environment, even if you pass <code>"deploy-modules": false</code> .	<code>true</code> , <code>false</code>
"modules"	Specifies the modules to be deployed to the selected environments.	Accepts a comma- and space-separated list of modules to deploy.

Key	Definition	Values
"wait"	<p>Specifies whether Code Manager must wait for all deploys to finish or error before it returns a response.</p> <p>If specified <code>true</code>, Code Manager returns a response after all deploys have either finished or errored. If specified <code>false</code> or not specified, returns a response showing that deploys are queued.</p> <p>Specify together with the "deploy-all" or "environment" keys.</p>	<code>true</code> , <code>false</code>
"dry-run"	<p>Tests the Code Manager connection to each of the configured remotes and attempts to fetch a list of environments from each, reporting any connection errors.</p> <p>Specify together with the "deploy-all" or "environment" keys.</p>	<code>true</code> , <code>false</code>

For example, to deploy the production and testing environments and return results after the deployments complete or fail, pass the "environments" key with the "wait" key:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{ "environments": [ "production", "testing" ], "wait": true }'

curl --header "$type_header" --header "$auth_header" --cacert "$cacert" --
request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

To deploy all configured environments and return only the queuing results, pass the "deploy-all" key without the "wait" key:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys"
data='{ "deploy-all": true }'

curl --header "$type_header" --header "$auth_header" --cacert "$cacert" --
request POST "$uri" --data "$data"
```

Response format

The `POST /deploys` response contains a list of objects, each containing data about a queued or deployed environment.

If the request did not include a `"wait"` key, the response returns each environment's name and status.

If the `"wait"` key was included in the request, the response returns information about each deployment. This information includes a `deploy-signature`, which is the commit SHA of the control repo that Code Manager used to deploy the environment. The output also includes two other SHAs that indicate that file sync is aware that the environment has been deployed to the code staging directory.

The response shows any deployment failures, even after they have successfully deployed. The failures remain in the response until cleaned up by garbage collection.

Key	Description	Value
<code>"environment"</code>	The name of the environment queued or deployed.	A string specifying the name of an environment.
<code>"id"</code>	Identifies the queue order of the code deploy request.	An integer generated by Code Manager.
<code>"status"</code>	The status of the code deployment for that environment.	Can be one of the following: <ul style="list-style-type: none"> <code>"new"</code>: The deploy request has been accepted but not yet queued. <code>"complete"</code>: The deploy is complete and has been synced to the live code directory on primary server and compilers. <code>"failed"</code>: The deploy failed. <code>"queued"</code>: The deploy is queued and waiting.
<code>"deploy-signature"</code>	The commit SHA of the control repo that Code Manager used to deploy code in that environment.	A Git SHA.
<code>"file-sync"</code>	Commit SHAs used internally by file sync to identify the code synced to the code staging directory	Git SHAs for the <code>"environment-commit"</code> and <code>"code-commit"</code> keys.

For example, for a `/deploys` request without the `"wait"` key, the response shows only `"new"` or `"queued"` status, because this response is returned as soon as Code Manager accepts the request.

```
[
  {
    "environment": "production",
    "id": 1,
    "status": "queued"
  },
  {
    "environment": "testing",
    "id": 2,
    "status": "queued"
  }
]
```

If you pass the "wait" key with your request, Code Manager doesn't return any response until the environment deployments have either failed or completed. For example:

```
[
  {
    "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
    "file-sync": {
      "environment-commit": "6939889b679fdb1449545c44f26aa06174d25c21",
      "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1"
    },
    "environment": "production",
    "id": 3, Code Manager
    "status": "complete"
  }
]
```

Error responses

If any deployments fail, the response includes an error object for each failed environment. Code Manager returns deployment results only if you passed the "wait" key; otherwise, it returns queue information.

Key	Definition	Value
"environment"	The name of the environment queued or deployed.	A string specifying the name of an environment.
"error"	Information about the deployment failure.	Contains keys with error details.
"corrected-name"	The name of the environment.	A string specifying the name.
"kind"	The kind of error encountered.	An error type.
"msg"	The error message.	An error message.
"id"	Identifies the queue order of the code deploy request.	An integer generated by Code Manager.
"status"	The status of the requested code deployment.	For errors, status is "failed".

For example, information for a failed deploy of an environment might look like:

```
{
  "environment": "test14",
  "error": {
    "details": {
      "corrected-name": "test14"
    },
    "kind": "puppetlabs.code-manager/deploy-failure",
    "msg": "Errors while deploying environment 'test14' (exit code: 1):\nERROR\t -> Authentication failed for Git remote \"https://github.com/puppetlabs/puffppetlabs-apache\".\n"
  },
  "id": 52,
  "status": "failed"
}
```


POST /v1/webhook

Use the Code Manager `/webhook` endpoint to trigger code deploys based on your control repo updates.

Request format

The `POST /webhook` request consists of a specially formatted URL that specifies the webhook type, an optional branch prefix, and a PE authentication token.

You must pass the authentication token with the `token` query parameter. To use a GitHub webhook with the Puppet signed certificate, you must disable SSL verification.

In your URL, include:

- The name of the primary server (for example, `primary.example.com`)
- The Code Manager port (for example, `8170`)
- The endpoint (`/v1/webhook`)
- The `type` parameter with a valid value, such as `gitlab`.
- The `prefix` parameter and value.
- The `token` parameter with the complete token.

Parameter	Definition	Values
<code>type</code>	Required. Specifies what type of POST body to expect.	<ul style="list-style-type: none"> • <code>github</code> • <code>gitlab</code> • <code>tfs-git</code> • <code>bitbucket</code> • <code>stash</code>
<code>prefix</code>	Optional. Specifies a prefix to use in translating branch names to environments.	A string specifying a branch prefix.
<code>token</code>	Specifies the PE authorization token to use. Required unless you disable <code>webhook_authentication</code> in Code Manager configuration.	The complete authentication token.

For example, a GitHub webhook might look like this:

```
https://primary.example.com:8170/code-manager/v1/webhook?type=github&token=$TOKEN
```

A Stash webhook with the optional `prefix` parameter specified might look like:

```
https://primary.example.com/code-manager/v1/webhook?type=stash&prefix=dev&token=$TOKEN
```

You must attach the complete authentication token to the request. A GitHub request with the entire token attached might look like this:

```
https://primary.example.com:8170/code-manager/v1/webhook?type=github&token=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkZXBs3kiLCJpYXQiOiJFOTBGRC0r2J87Pj2sDsyZ-EMK-7sZalBTswy2e3uSv1Z6ulXaIQQd69PQnSSBExQotExDgXZInyQa_le2gwTo4smjUaBd6_lnPyr6GJ4hjB4-ft8dNnVuvZae5WPkKt-sNJKJwE9LiEd4W42aCYfse1KNgPuXR5m77SRsUy86ymthVPKHqqexEyuS7LGeQJvyQE1qEe jSdbiLg6zn1JXhg1W
```

```
NrE17oxrrNkU0ZxioUgDeqGycwvNIaMazztM9NyD-
dWmZc4dKJsqm0su0CRkMSWcYPPaeIcsYFI7XSaeC65N4RLIKhUfwIxxE-
uODEhcl3mTr9rwZGnVMu3WrY7t6wlbDM9FomPejGM2aJoZ05PinAIYvX3oH5QJ9fam0pVLb-
mI3bvKkm2wKAgpc4Dhlut9sqLWkG8-
AwMA4r_oEvLwFzf8clzk34zNyPG7BvlnPle99HjQues690L-
fknSdFiXyRZeRThvZop0SWJzvUSR49etmk-
OxnMbQE4tCBWZr_khEG5jUDzeKt3PIiXdxmUaaEPHzo6Vl9XIY5
```

Response format

When you trigger your webhook by pushing changes to your control repository, you can see the response in your Git provider interface. Code Manager does not give a command line response to typical webhook usage.

If you hit the webhook endpoint directly using a curl command with a properly formatted request, valid authentication token, and a valid value for the "type" parameter, Code Manager returns a `{ "status": "OK" }` response, whether or not code successfully deployed.

Error responses

When you trigger your webhook by pushing changes to your control repository, you can see any errors in your Git provider interface. Code Manager does not give a command line response to typical webhook usage.

If you hit the webhook endpoint directly using a curl command with a request that has an incorrect "type" value or no "type" value, you get an error response such as:

```
{ "kind": "puppetlabs.code-manager/unrecognized-webhook-
type", "msg": "Unrecognized webhook type: 'githubby'", "details": "Currently
supported valid webhook types include: github gitlab stash tfs-git
bitbucket" }
```

GET /v1/deploys/status

The Code Manager `/deploys/status` endpoint returns the status of all code deployments that Code Manager is processing for each environment. You can specify an `id` query parameter to return the status of a particular deployment.

Request format

The body of the `GET /deploys/status` request must include the authentication token and a URL containing:

- The name of the primary server, such as `primary.example.com`.
- The Code Manager port, such as 8170.
- The endpoint.

To return the status of all code deployments:

```
type_header='Content-Type: application/json'
auth_header='X-Authentication: $(puppet-access show)'
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/
status"

curl --insecure --header "$type_header" --header "$auth_header" --header
"$type_header" "$uri"
```

To return the status of a specified code deployment:

```
type_header='Content-Type: application/json'
auth_header='X-Authentication: $(puppet-access show)'
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/
status?id=1"
```

```
curl --insecure --header "$type_header" --header "$auth_header" --header
"$type_header" "$uri"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

Response format

The `/deploys/status` endpoint responds with a list of all code deployment requests that Code Manager is processing.

The response contains three sections:

- `"deploys-status"`: Lists the status for each code deployment that Code Manager is processing, including failed deployments. Deployments can be `"new"`, `"queued"`, `"deploying"`, or `"failed"`. Environments that have been successfully deployed to either the code staging or live code directories are displayed in the `"file-sync-storage-status"` or `"file-sync-client-status"` sections, respectively.
- `"file-sync-storage-status"`: Lists all environments that Code Manager has successfully deployed to the code staging directory, but not yet synced to the live code directory.
- `"file-sync-client-status"`: Lists status for your primary server and each compiler that Code Manager is deploying environments to, including whether the code in the primary server's staging directory has been synced to its live code directory.

The response can contain the following keys:

Key	Definition	Values
"queued"	The environment is queued for deployment.	For each deployment: <ul style="list-style-type: none"> • "id" • "environment" • "queued-at"
"deploying"	The environment is in the process of being deployed.	For each deployment: <ul style="list-style-type: none"> • "id" • "environment" • "queued-at"
"new"	Code Manager has accepted a request to deploy the environment, but has not yet queued it.	For each deployment: <ul style="list-style-type: none"> • "id" • "environment" • "queued-at"

Key	Definition	Values
"failed"	The code deployment for the environment has failed.	For each deployment: <ul style="list-style-type: none"> • "id" • "environment" • "error" • "details" • "corrected-name" • "kind" • "msg" • "queued-at"
"deployed"	Lists information for each deployed environment.	For each deployment: <ul style="list-style-type: none"> • "environment" • "date" • "deploy-signature"
"all-synced"	Whether all requested code deployments have been synced to the live code directories on their respective servers.	true, false
"file-sync-clients"	List of all servers that Code Manager is deploying code to.	For each deployment listed: <ul style="list-style-type: none"> • "last_check_in_time" • "synced-with-file-sync-storage" • "deployed" • "environment" • "date" • "deploy-signature"

For example, a complete response without an ID specified might look like this:

```
{
  "deploys-status": {
    "queued": [
      {
        "environment": "dev",
        "id": 3,
        "queued-at": "2018-05-15T17:42:34.988Z"
      }
    ]
  }
}
```

```

],
"deploying":[
  {
    "environment":"test",
    "id":1,
    "queued-at":"2018-05-15T17:42:34.988Z"
  },
  {
    "environment":"prod",
    "id":2,
    "queued-at":"2018-05-15T17:42:34.988Z"
  }
],
"new":[
],
"failed":[
]
},
"file-sync-storage-status":{
  "deployed":[
    {
      "environment":"prod",
      "date":"2018-05-10T21:44:24.000Z",
      "deploy-signature":"66d620604c9465b464a3dac4884f96c43748b2c5"
    },
    {
      "environment":"test",
      "date":"2018-05-10T21:44:25.000Z",
      "deploy-signature":"24ecc7bac8a4d727d6a3a2350b6fda53812ee86f"
    },
    {
      "environment":"dev",
      "date":"2018-05-10T21:44:21.000Z",
      "deploy-signature":"503a335c99a190501456194d13ff722194e55613"
    }
  ]
},
"file-sync-client-status":{
  "all-synced":false,
  "file-sync-clients":{
    "chihuahua":{
      "last_check_in_time":null,
      "synced-with-file-sync-storage":false,
      "deployed":[
    ]
      },
    "localhost":{
      "last_check_in_time":"2018-05-11T22:41:20.270Z",
      "synced-with-file-sync-storage":true,
      "deployed":[
        {
          "environment":"prod",
          "date":"2018-05-11T22:40:48.000Z",
          "deploy-
signature":"66d620604c9465b464a3dac4884f96c43748b2c5"
        },
        {
          "environment":"test",
          "date":"2018-05-11T22:40:48.000Z",
          "deploy-
signature":"24ecc7bac8a4d727d6a3a2350b6fda53812ee86f"

```

```

    },
    {
      "environment": "dev",
      "date": "2018-05-11T22:40:50.000Z",
      "deploy-
signature": "503a335c99a190501456194d13ff722194e55613"
    }
  ]
}
}
}
}

```

The `/deploys/status?id` endpoint responds with details about a specified code deployment.

The response contains three sections:

- `"environment"` or `"project"`: Type of entity being deployed.
- `"status"`: Status of the deployment, including `"queued"`, `"deploying"`, `"new"`, `"failed"`, `"syncing"`, or `"synced"`.
- `"queued-at"`: Timestamp specifying when the deployment was first queued.

For example, a complete response with an ID specified might look like this:

```

{
  "environment": "production",
  "id": 1,
  "status": "deploying",
  "queued-at": "2018-05-10T21:44:25.000Z"
}

```

Error responses

If code deployment fails, the `"deploys-status"` section of the response provides an `"error"` key for the environment with the failure. The `"error"` key contains information about the failure.

Deployment failures can remain in the response even after the environment in question is successfully deployed. Old failures are removed from the `"deploys-status"` response during garbage collection.

Key	Definition	Value
<code>"environment"</code>	The name of the environment queued or deployed.	A string specifying the name of an environment.
<code>"error"</code>	Information about the deployment failure.	Contains keys with error details.
<code>"corrected-name"</code>	The name of the environment.	A string specifying the name.
<code>"kind"</code>	The kind of error encountered.	An error type.
<code>"msg"</code>	The error message.	An error message.
<code>"queued-at"</code>	The date and time when the environment was queued.	A date and time stamp.

For example, with a failure, the `"deploys-status"` response section might look something like:

```

[
  {
    "deploys-status": {
      "deploying": [],
      "failed": [

```

```

    {
      "environment": "test14",
      "error": {
        "details": {
          "corrected-name": "test14"
        },
        "kind": "puppetlabs.code-manager/deploy-failure",
        "msg": "Errors while deploying environment
'test14' (exit code: 1):\nERROR\t -> Authentication failed for Git remote
'https://github.com/puppetlabs/puffppetlabs-apache'\n.\n"
      },
      "queued-at": "2018-06-01T21:28:18.292Z"
    }
  ],
  "new": [],
  "queued": []
},
...
}
]

```

Managing code with r10k

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository.

Based on the code in your control repo branches, r10k creates environments on your primary server and installs and updates the modules you want in each environment. If you are running PE 2015.3 or later, we encourage you to use Code Manager, which uses r10k in the background to automate the deployment of your code. If you use Code Manager, you won't need to manage or interact with r10k manually. To learn more about Code Manager and begin setup, see the Code Manager page. However, if you're already using r10k and aren't ready to switch to Code Manager, you can continue using r10k alone.

To set up r10k to manage your environments:

1. Set up a control repository for your code.
2. Create a Puppetfile to manage the content for your environment.
3. Configure r10k. Optionally, you can also customize your r10k configuration in Hiera.
4. Run r10k to deploy your environments and modules.

We've also included a reference of r10k subcommands.

- [Configure r10k](#) on page 664

To configure r10k in an existing PE installation, set r10k parameters in the console. You can also adjust r10k settings in the console.

- [Customizing r10k configuration](#) on page 664

If you need a customized r10k configuration, you can set specific parameters with Hiera.

- [Deploying environments with r10k](#) on page 671

Deploy environments on the command line with the `r10k` command.

- [r10k command reference](#) on page 673

r10k accepts several command line actions, with options and subcommands.

Configure r10k

To configure r10k in an existing PE installation, set r10k parameters in the console. You can also adjust r10k settings in the console.

Before you begin

Ensure that you have a Puppetfile and a control repo. You also need the SSH private key that you created when you made your control repo.

1. In the console, set the following parameters in the `puppet_enterprise::profile::master` class in the **PE Master** node group:

- `r10k_remote`

This is the location of your control repository. Enter a string that is a valid URL for your Git control repository, such as `"git@<YOUR.GIT.SERVER.COM>:puppet/control.git"`.

- `r10k_private_key`

This is the path to the private key that permits the `pe-puppet` user to access your Git repositories. This file must be located on the primary server, owned by the `pe-puppet` user, and in a directory that the `pe-puppet` user has permission to view. We recommend `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`. Enter a string, such as `" /etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"`.

2. Run Puppet on your primary server and compilers.

You can now customize your r10k configuration in Hiera, if needed. After r10k is configured, you can deploy your environments from the command line. PE does not automatically run r10k at the end of installation.

Related information

[Configuration parameters and the `pe.conf` file](#) on page 103

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

Customizing r10k configuration

If you need a customized r10k configuration, you can set specific parameters with Hiera.

To customize your configuration, add keys to your control repository hierarchy in the `data/common.yaml` file in the format `pe_r10k::<parameter>`.

In this example, the first parameter specifies where to store the cache of your Git repos, and the second sets where the source repository should be fetched from.

```
pe_r10k::cachedir: /var/cache/r10k
pe_r10k::remote: git://git-server.site/my-org/main-modules
```

Remember: After changing any of these settings, run r10k on the command line to deploy your environments. PE does not automatically run r10k at the end of installation.

Related information

[Configure settings with Hiera](#) on page 161

Hiera is a hierarchy-based method of configuration management that relies on a “defaults, with overrides” system. When you add a parameter or setting to your Hiera data, Hiera searches through the data in the order it was written to find the value you want to change. Once found, it overrides the default value with the new parameter or setting. You can use Hiera to manage your PE configuration settings.

Add r10k parameters in Hiera

To customize your r10k configuration, add parameters to your control repository hierarchy in the `data/common.yaml` file.

1. Add the parameter in the `data/common.yaml` file in the format `pe_r10k::<parameter>`.
2. Run Puppet on the primary server to apply changes.

Configuring Forge settings

To configure how r10k downloads modules from the Forge, specify `forge_settings` in Hiera.

This parameter configures where Forge modules should be installed from, and sets a proxy for all Forge interactions. The `forge_settings` parameter accepts a hash with the following values:

- `baseurl`
- `proxy`

baseurl

Indicates where Forge modules should be installed from. Defaults to `https://forgeapi.puppetlabs.com`.

```
pe_r10k::forge_settings:
  baseurl: 'https://private-forge.mysite'
```

proxy

Sets the proxy for all Forge interactions.

This setting overrides the global `proxy` setting on Forge operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. See the global `proxy` setting for more information and examples.

```
proxy: 'http://proxy.example.com:3128'
```

Configuring purge levels

The `purge_levels` setting controls which unmanaged content r10k purges during a deployment.

This setting accepts an array of strings specifying what content r10k purges during code deployments. Available values are:

- `deployment`
- `environment`
- `puppetfile`
- `purge_whitelist`

The default value for this setting is `['deployment', 'puppetfile']`. With these values, r10k purges:

- Any content that is not managed by the sources declared in the `remote` or `sources` settings.
- Any content that is not declared in the Puppetfile, but is found in a directory managed by the Puppetfile.

```
deploy:
  purge_levels: [ 'deployment', 'environment', 'puppetfile' ]
```

deployment

After each deployment, in the configured basedir, r10k recursively removes content that is not managed by any of the sources declared in the `remote` or `sources` parameters.

Disabling this level of purging could cause the number of deployed environments to grow without bound, because deleting branches from a control repo would no longer cause the matching environment to be purged.

environment

After a given environment is deployed, r10k recursively removes content that is neither committed to the control repo branch that maps to that environment, nor declared in a Puppetfile committed to that branch.

With this purge level, r10k loads and parses the Puppetfile for the environment even if the `--puppetfile` flag is not set. This allows r10k to check whether or not content is declared in the Puppetfile. However, Puppetfile content is deployed only if the environment is new or the `--puppetfile` flag is set.

If r10k encounters an error while evaluating the Puppetfile or deploying its contents, no environment-level content is purged.

puppetfile

After Puppetfile content for a given environment is deployed, r10k recursively removes content in any directory managed by the Puppetfile, if that content is not also declared in that Puppetfile.

Directories considered to be managed by a Puppetfile include the configured `moduledir` (which defaults to "modules"), as well as any alternate directories specified as an `install_path` option to any Puppetfile content declarations.

purge_whitelist

The `purge_whitelist` setting exempts the specified filename patterns from being purged.

This setting affects environment level purging only. Any given value must be a list of shell style filename patterns in string format.

See the [Ruby documentation](#) for the `fnmatch` method for more details on valid patterns. Both the `FNM_PATHNAME` and `FNM_DOTMATCH` flags are in effect when r10k considers the whitelist.

Patterns are relative to the root of the environment being purged and, by default, **do not match recursively**. For example, a whitelist value of `*myfile*` would preserve only a matching file at the root of the environment. To preserve the file throughout the deployed environment, you would need to use a recursive pattern such as `**/*myfile*`.

Files matching a whitelist pattern might still be removed if they exist in a folder that is otherwise subject to purging. In this case, use an additional whitelist rule to preserve the containing folder.

```
deploy:
  purge_whitelist: [ 'custom.json', '**/*.xpp' ]
```

Locking r10k deployments

The `deploy: write_lock` setting allows you to temporarily disallow r10k code deploys without completely removing the r10k configuration.

This `deploy` subsetting is useful to prevent r10k deployments at certain times or to prevent deployments from interfering with a common set of code that might be touched by multiple r10k configurations.

```
deploy:
  write_lock: "Deploying code is disallowed until the next maintenance
window."
```

Configuring sources

If you are managing more than one repository with `r10k`, specify a map of your source repositories.

Use the `source` parameter to specify a map of sources. Configure this setting if you are managing more than just Puppet environments, such as when you are also managing Hiera data in its own control repository.

To use multiple control repos, the `sources` setting and the `repositories` setting must match.

If `sources` is set, you cannot use the global `remote` and `r10k_basedir` settings. Note that `r10k` raises an error if different sources collide in a single base directory. If you are using multiple sources, use the `prefix` option to prevent collisions.

This setting accepts a hash with the following subsettings:

- `remote`
- `basedir`
- `prefix`
- `invalid_branches`

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  basedir: "/etc/puppetlabs/puppet/environments"
  prefix: true
  invalid_branches: 'error'
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  basedir: "/etc/puppetlabs/puppet/environments"
  prefix: "testing"
  invalid_branches: 'correct_and_warn'
```

remote

Specifies where the source repository should be fetched from. The remote must be able to be fetched without any interactive input. That is, you cannot be prompted for usernames or passwords in order to fetch the remote.

Accepts a string that is any valid URL that `r10k` can clone, such as:

```
git://git-server.site/my-org/main-modules
```

basedir

Specifies the path where environments are created for this source. This directory is entirely managed by `r10k`, and any contents that `r10k` did not put there are removed.

Note that the `basedir` setting **must match** the `environmentpath` in your `puppet.conf` file, or Puppet won't be able to access your new directory environments.

prefix

Allows environment names to be prefixed with this string. Alternatively, if `prefix` is set to `true`, the source's name is used. This prevents collisions when multiple sources are deployed into the same directory.

In the `sources` example above, two "main-modules" environments are set up in the same base directory. The `prefix` setting is used to differentiate the environments: the first is named "myorg-main-modules", and the second is "testing-main-modules".

ignore_branch_prefixes

Causes branches from a source which match any of the prefixes listed in the setting to be ignored. The match can be full or partial. On deploy, each branch in the repo has its name tested against all prefixes listed in `ignore_branch_prefixes` and, if the prefix is found, then an environment is not deployed for this branch.

The setting is a list of strings. In the following example, branches in "mysource" with the prefixes "test" and "dev" are not deployed.

```
---
sources:
  mysource:
    basedir: '/etc/puppet/environments'
    ignore_branch_prefixes:
      - 'test'
      - 'dev'
```

If `ignore_branch_prefixes` is not specified, then all branches are deployed.

invalid_branches

Specifies how branch names that cannot be cleanly mapped to Puppet environments are handled. This option accepts three values:

- 'correct_and_warn' is the default value and replaces non-word characters with underscores and issues a warning.
- 'correct' replaces non-word characters with underscores without warning.
- 'error' ignores branches with non-word characters and issues an error.

Configuring the r10k base directory

The `r10k` base directory specifies the path where environments are created for this source.

This directory is entirely managed by `r10k`, and any contents that `r10k` did not put there are removed. If `r10k_basedir` is not set, it uses the default `environmentpath` in your `puppet.conf` file. If `r10k_basedir` is set, you cannot set the `sources` parameter.

Note that the `r10k_basedir` setting **must match** the `environmentpath` in your `puppet.conf` file, or Puppet won't be able to access your new directory environments.

Accepts a string, such as: `/etc/puppetlabs/code/environments`.

Configuring r10k Git settings

To configure `r10k` to use a specific Git provider, a private key, a proxy, or multiple repositories with Git, specify the `git_settings` parameter.

The `r10k git_settings` parameter accepts a hash of the following settings:

- `private_key`
- `proxy`
- `repositories`
- `provider`

Contains settings specific to Git. Accepts a hash of values, such as:

```
pe_r10k::git_settings:
  provider: "rugged"
  private_key: "/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa"
  username: "git"
```

private_key

Specifies the file containing the default private key used to access control repositories, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.rsa`. This file must have read permissions for the `pe-puppet` user. The SSH key cannot require a password. This setting is required, but by default, the value is supplied in the `puppet_enterprise::profile::master` class.

proxy

Sets a proxy specifically for Git operations that use an HTTP(S) transport.

This setting overrides the global `proxy` setting on Git operations only. You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. To set a proxy for a specific Git repository only, set `proxy` in the `repositories` subsetting of `git_settings`. See the global `proxy` setting for more information and examples.

```
proxy: 'http://proxy.example.com:3128'
```

repositories

Specifies a list of repositories and their respective private keys. Use this setting if you want to use multiple control repos.

To use multiple control repos, the `sources` setting and the `repositories` setting must match. Accepts an array of hashes with the following keys:

- `remote`: The repository these settings should apply to.
- `private_key`: The file containing the private key to use for this repository. This file must have read permissions for the `pe-puppet` user. This setting overrides the global `private_key` setting.
- `proxy`: The proxy setting allows you to set or override the global proxy setting for a single, specific repository. See the global `proxy` setting for more information and examples.

```
pe_r10k::git_settings:
  repositories:
    - remote: "ssh://tessier-ashpool.freeside/protected-repo.git"
      private_key: "/etc/puppetlabs/r10k/ssh/id_rsa-protected-repo-deploy-key"
    - remote: "https://git.example.com/my-repo.git"
      proxy: "https://proxy.example.com:3128"
```

username

Optionally, specify a username when the Git remote URL does not provide a username.

Configuring proxies

To configure proxy servers, use the `proxy` setting. You can set a global proxy for all HTTP(S) operations, for all Git or Forge operations, or for a specific Git repository only.

proxy

To set a proxy for all operations occurring over an HTTP(S) transport, set the global `proxy` setting. You can also set an authenticated proxy with either Basic or Digest authentication.

To override this setting for Git or Forge operations only, set the `proxy` subsetting under the `git_settings` or `forge_settings` parameters. To override for a specific Git repository, set a proxy in the `repositories` list of `git_settings`. To override this setting with no proxy for Git, Forge, or a particular repository, set that specific `proxy` setting to an empty string.

In this example, the first proxy is set up without authentication, and the second proxy uses authentication:

```
proxy: 'http://proxy.example.com:3128'
proxy: 'http://user:password@proxy.example.com:3128'
```

r10k proxy defaults and logging

By default, r10k looks for and uses the first environment variable it finds in this list:

- HTTPS_PROXY
- https_proxy
- HTTP_PROXY
- http_proxy

If no proxy setting is found in the environment, the global proxy setting defaults to no proxy.

The proxy server used is logged at the "debug" level when r10k runs.

Configuring postrun commands

To configure a command to be run after all deployment is complete, add the postrun parameter.

The postrun optional command to be run after r10k finishes deploying environments. The command must be an array of strings to be used as an argument vector. You can set this parameter only one time.

```
postrun: [ '/usr/bin/curl', '-F', 'deploy=done', 'http://my-app.site/endpoint' ]
```

r10k parameters

The following parameters are available for r10k. Parameters are optional unless otherwise stated.

Parameter	Description	Value	Default
cachedir	Specifies the path to the directory where you want the cache of your Git repos to be stored.	A valid directory path	/var/cache/r10k
deploy	Controls how r10k code deployments behave. See Configuring purge levels and Locking deployments for usage details.	Accepts settings for: <ul style="list-style-type: none"> • purge_level • write_lock 	<ul style="list-style-type: none"> • purge_level: ['deployment', 'puppetfile'] • write_lock: Not set by default .
forge_settings	Contains settings for downloading modules from the Puppet Forge. See Configuring Forge settings for usage details.	Accepts a hash of: <ul style="list-style-type: none"> • baseurl • proxy 	No default.
git_settings	Configures Git settings: Git provider, private key, proxies, and repositories. See Configuring Git settings for usage details.	Accepts a hash of: <ul style="list-style-type: none"> • provider • private_key • proxy • repositories 	No default.
proxy	Configures a proxy server to use for all operations that occur over an HTTP(S) transport. See Configuring proxies for usage details.	Accepts a string specifying a URL to proxy server, without authentication, or with Basic or Digest authentication.	No default set.

Parameter	Description	Value	Default
<code>postrun</code>	An optional command to be run after <code>r10k</code> finishes deploying environments.	An array of strings to use as an argument vector.	No default set.
<code>remote</code>	Specifies where the source repository should be fetched from. The remote cannot require any prompts, such as for usernames or passwords. If <code>remote</code> is set, you cannot use sources.	Accepts a string that is any valid SSH URL that <code>r10k</code> can clone.	By default, uses the <code>r10k_remote</code> value set in the console.
<code>r10k_basedir</code>	Specifies the path where environments are created for this source. See Configuring the base directory for usage details.	A valid file path, which must match the value of <code>environmentpath</code> in your <code>puppet.conf</code> file.	Uses the value of the <code>environmentpath</code> in your <code>puppet.conf</code> file.
<code>sources</code>	Specifies a map of sources to be passed to <code>r10k</code> . Use if you are managing more than just Puppet environments. See Configuring sources for usage details.	A hash of: <ul style="list-style-type: none"> <code>basedir</code> <code>remote</code> <code>prefix</code> <code>ignore_branch_prefixes</code> 	No default.

Deploying environments with `r10k`

Deploy environments on the command line with the `r10k` command.

The first time you run `r10k`, deploy all environments and modules by running `r10k deploy environment`.

This command:

1. Checks your control repository to see what branches are present.
2. Maps those branches to the Puppet directory environments.
3. Clones your Git repo, and either creates (if this is your first run) or updates (if it is a subsequent run) your directory environments with the contents of your repo branches.

Note:

When running commands to deploy code on your primary server, `r10k` needs write access to your environment path. Run `r10k` as the `pe-puppet` user or as root. Running the user as root requires access control to the root user.

Updating environments

To update environments with `r10k`, use the `deploy environment` command.

Updating all environments

The `deploy environment` command updates all existing environments and recursively creates new environments.

This command updates modules only on the first deployment of a given environment. On subsequent updates, it only updates the environment.

From the command line, run `r10k deploy environment`

Updating all environments and modules

With the `--puppetfile` flag, the `deploy environment` command updates all environments and modules.

This command:

- Updates all sources.
- Creates new environments.
- Deletes old environments.
- Recursively updates all environment modules specified in each environment's Puppetfile.

This command does the maximum possible work, and is therefore the slowest method for r10k deployments. Usually, you should use the less resource-intensive commands for updating environments and modules.

From the command line, run `r10k deploy environment --puppetfile`

Updating a single environment

To update a single environment, specify the environment name with the `deploy environment` command.

This command updates modules only on the first deployment of the specified environment. On subsequent updates, it only updates the environment.

If you are actively developing on a given environment, running the following command is the quickest way to deploy your changes:

```
r10k deploy environment <MY_WORKING_ENVIRONMENT>
```

Updating a single environment and its modules

To update both a single given environment and all of the modules contained in that environment's Puppetfile, add the `--puppetfile` flag to your command. This is useful if you want to make sure that a given environment is fully up to date.

On the command line, run `r10k deploy environment <MY_WORKING_ENVIRONMENT> --puppetfile`

Installing and updating modules

To update modules, use the `r10k deploy module` subcommand. This command installs or updates the modules specified in each environment's Puppetfile.

If the specified module is not described in a given environment's Puppetfile, that environment is skipped.

Updating specific modules across all environments

To update specific modules across all environments, specify the modules with the `deploy module` command.

Important: You must run the `r10k deploy environment` command without the `--puppetfile` flag before you update modules across all environments. This command fetches any updates to the applicable Puppetfile.

You can specify a single module or multiple modules. For example, to update the `apache`, `jenkins`, and `java` modules, run `r10k deploy module apache jenkins java`

Updating one or more modules in a single environment

To update specific modules in a single environment, specify both the environment and the modules.

Important: You must run the `r10k deploy <environment>` command without the `--puppetfile` flag before you update modules in a single environment. This command fetches any updates to the applicable Puppetfile.

On the command line, run `r10k deploy module` with:

- The environment option `-e`.
- The name of the environment.
- The names of the modules you want to update in that environment.

For example, to install the `apache`, `jenkins`, and `java` modules in the production environment, run `r10k deploy module -e production apache jenkins java`

Viewing environments with r10k

Display information about your environments and modules with the `r10k deploy display` subcommand. This subcommand does not deploy environments, but only displays information about the environments and modules `r10k` is managing.

This command can return varying levels of detail about the environments.

- To display all environments being managed by `r10k`, use the `deploy display` command. From the command line, run `r10k deploy display`
- To display all managed environments and Puppetfile modules, use the Puppetfile flag, `-p`. From the command line, run `r10k deploy display -p`
- To get detailed information about module versions, use the `-p` (Puppetfile) and `--detail` flags. This provides both the expected and actual versions of the modules listed in each environment's Puppetfile. From the command line, run `r10k deploy display -p --detail`
- To get detailed information about the modules only in specific environments, limit your query with the environment names. This provides both the expected and actual versions of the modules listed in the Puppetfile in each specified environment. For example, to see details on the modules for environments called `production`, `vmwr`, and `webrefactor`, run:

```
r10k deploy display -p --detail production vmwr webrefactor
```

r10k command reference

`r10k` accepts several command line actions, with options and subcommands.

r10k command actions

`r10k` includes several command line actions.

r10k command	Action
<code>deploy</code>	Performs operations on environments. Accepts <code>deploy</code> subcommands listed below.
<code>help</code>	Displays the <code>r10k</code> help page in the terminal.
<code>puppetfile</code>	Performs operations on a Puppetfile. Accepts <code>puppetfile</code> subcommands.
<code>version</code>	Displays your <code>r10k</code> version in the terminal.

```
r10k deploy
```

r10k command options

`r10k` commands accept the following options.

r10k command options	Action
<code>--color</code>	Enables color coded log messages.
<code>--help, -h</code>	Shows help for this command.
<code>--trace, -t</code>	Displays stack traces on application crash.
<code>--verbose, -v</code>	Sets log verbosity. Valid values: fatal, error, warn, notice, info, debug, debug1, debug2.

r10k deploy subcommands

You can use the following subcommands with the `r10k deploy` command.

```
r10k deploy --display
```

deploy subcommand	Action
<code>display</code>	Displays a list of environments in your deployment.
<code>display -p</code>	Displays a list of modules in the Puppetfile.
<code>display --detail</code>	Displays detailed information.
<code>display --fetch</code>	Update the environment sources. Allows you to check for missing environments.
<code>environment</code>	Deploys environments and their specified modules.
<code>environment -p</code>	Updates modules specified in the environment's Puppetfile.
<code>module</code>	Deploys a module in all environments.
<code>module -e</code>	Updates all modules in a particular environment.
<code>no-force</code>	Prevents the overwriting of local changes to Git-based modules.

See the related topic about deploying environments with `r10k` for examples of `r10k deploy` command use.

r10k puppetfile subcommands

The `r10k puppetfile` command accepts several subcommands for managing modules.

```
r10k puppetfile install
```

These subcommands must be run as the user with write access to that environment's modules directory. These commands interact with the Puppetfile in the current working directory, so before running the subcommand, make sure you are in the directory of the Puppetfile you want to use.

puppetfile subcommand	Action
<code>check</code>	Verifies the Puppetfile syntax is correct.
<code>install</code>	Installs all modules from a Puppetfile.
<code>install --puppetfile</code>	Installs modules from a custom Puppetfile path.
<code>install --moduledir</code>	Installs modules from a Puppetfile to a custom module directory location.
<code>install --update_force</code>	Installs modules from a Puppetfile with a forced overwrite of local changes to Git-based modules.
<code>purge</code>	Purges unmanaged modules from a Puppetfile managed directory.

About file sync

File sync helps Code Manager keep your Puppet code synchronized across your primary server and compilers.

When triggered by a web endpoint, file sync takes changes from your working directory on your primary server and deploys the code to a live code directory. File sync then automatically deploys that code to all compilers.

In addition, file sync ensures that your Puppet code is deployed only when it is ready. These deployments ensure that your agents' code won't change during a run. File sync also triggers an environment cache flush when the deployment has finished, to ensure that new agent runs happen against the newly deployed Puppet code.

File sync works with Code Manager, so you typically won't need to do anything with file sync directly. If you want to know more about how file sync works, or you need to work with file sync directly for testing, this page provides additional information.

File sync terms

There are a few terms that are helpful when you are working with file sync or tools that use file sync.

Live code directory

This is the directory that all of the Puppet environments, manifests, and modules are stored in. This directory is used by Puppet Server for catalog compilation. It corresponds to the Puppet Server `master-code-dir` setting and the `Puppet$codedir` setting. The default value is `/etc/puppetlabs/code`. In file sync configuration, you might see this referred to simply as `live-dir`. This directory exists on your primary server and compilers.

Staging code directory

This is the directory where you stage your code changes before rolling them out to the live code dir. You can move files into this directory in your usual way. Then, when you trigger a file sync deployment, file sync moves the changes to the live code dir on your primary server and compilers. This directory exists only on the primary server; compilers do not need a staging directory. The default value is `/etc/puppetlabs/code-staging`.

How file sync works

File sync helps distribute your code to your primary server, compilers, and agents.

By default, file sync is disabled and the staging directory is not created on the primary server. If you're upgrading from 2015.2 or earlier, file sync is disabled after the upgrade. You must enable file sync, and then run Puppet on your primary server and compilers. This creates the staging directory on the primary server, which you can then populate with your Puppet code. File sync can then commit your code; that is, it can prepare the code for synchronization to the live code directory, and then your compilers. Normally, Code Manager triggers this commit automatically, but you can trigger a commit by hitting the file sync endpoint.

For example:

```
type_header='Content-Type: application/json'
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"
uri="https://$(puppet config print server):8140/file-sync/v1/commit"
data='{ "commit-all": true }'

curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
--request POST "$uri" --data "$data"
```

See [Usage notes for curl examples](#) for information about forming curl commands.

The above command is run from the primary server and contains the following variables:

- `fqdn`: Fully qualified domain name of the primary server.
- `cert`: The ssl cert for the primary server.
- `cacert`: The Puppet CA's certificate.
- `key`: The private key for the primary server.

This command commits all of the changes in the staging directory. After the commit, when any compilers check the file sync service for changes, they receive the new code and deploy it into their own code directories, where it is available for agents checking in to those compilers. (By default, compilers check file sync every 5 seconds.)

Commits can be restricted to a specific environment and can include details such as a message, and information about the commit author.

Enabling or disabling file sync

File sync is normally enabled or disabled automatically along with Code Manager.

File sync's behavior is linked to that of Code Manager. Because Code Manager is disabled by default, file sync is also disabled. To enable file sync, enable Code Manager. You can enable and configure Code Manager either during or after PE installation.

The `file_sync_enabled` parameter in the `puppet_enterprise::profile::master` class in the console defaults to `automatic`, which means that file sync is enabled and disabled automatically with Code Manager. If you set this parameter to `true`, it forces file sync to be enabled even if Code Manager is disabled. The `file_sync_enabled` parameter doesn't appear in the class definitions --- you must add the parameter to the class in order to set it.

Resetting file sync

If file sync has entered a failure state, consumed all available disk space, or a repository has become irreparably corrupted, reset the service.

Resetting deletes the commit history for all repositories managed by file sync, which frees up disk space and returns the service to a "fresh install" state while preserving any code in the staging directory.

1. On the primary server, perform the appropriate action:
 - If you use file sync with Code Manager, ensure that any code ready for deployment is present in the staging directory, and that the code most recently deployed is present in your control repository so that it can be re-synced.
 - If you use file sync with r10k, perform an r10k deploy and ensure that the code most recently deployed is present in your control repository so that it can be re-synced.
 - If you use file sync alone, ensure that any code ready for deployment is present in `/etc/puppetlabs/code-staging`.
2. Shut down the pe-puppetserver service: `puppet resource service pe-puppetserver ensure=stopped`.
3. Delete the data directory located at `/opt/puppetlabs/server/data/puppetserver/filesync/storage`.
4. Restart the pe-puppetserver service: `puppet resource service pe-puppetserver ensure=running`.
5. Perform the appropriate action:
 - If you use file sync with Code Manager, use Code Manager to deploy all environments.
 - If you use file sync alone or with r10k, perform a commit.

File sync is now reset. The service creates fresh repositories on each client and the storage server for the code it manages.

Checking your deployments

You can manually look up information about file sync's deployments by using curl commands that hit the `status` endpoint.

To look up deployment information, run the following curl command:

```
curl --insecure "https://$(puppet config print server):8140/status/v1/services/file-sync-storage-service?level=debug"
```

The curl command returns a list of:

- All of the clients that file sync is aware of.
- When those clients last checked in.
- Which commit they have deployed.

Note: This command returns output in JSON, so you can pipe it to `python -m json.tool` for better readability. See [Usage notes for example commands](#) on page 24 for information about forming curl commands.

You can check if a specific commit was deployed by viewing the `latest_commit`. This commit is listed as the `latest_commit`.

Important: The SHA reported as `latest_commit` is internal to file sync and not from the user's control repository.

Cautions

There are a few things to be aware of with file sync.

Always use the staging directory

Always make changes to your Puppet code in the staging directory. If you have edited code in the live code directory on the primary server or any compilers, *it's overwritten* by file sync on the next commit.

The `enable-forceful-sync` parameter is set to `true` by default in PE. When `false`, file sync does not overwrite changes in the code directory, but instead logs errors in `/var/log/puppetlabs/puppetserver/puppetserver.log`. To set this parameter to `false`, add via Hiera (`puppet_enterprise::master::file_sync::file_sync_enable_forceful_sync: false`).

The puppet module command and file sync

The `puppet module` command doesn't work with file sync. If you are using file sync, specify modules in the Puppetfile and use Code Manager to handle your syncs.

Permissions

File sync runs as the `pe-puppet` user. To sync files, file sync **must** have permission to read the staging directory and to write to all files and directories in the live code directory. To make sure file sync can read and write what it needs to, ensure that these code directories are both owned by the `pe-puppet` user:

```
chown -R pe-puppet /etc/puppetlabs/code /etc/puppetlabs/code-staging
```

Environment isolation metadata

File sync generates `.pp` metadata files in both your live and staging code directories. These files provide environment isolation for your resource types, ensuring that each environment uses the correct version of the resource type. Do not delete or modify these files. Do not use expressions from these files in regular manifests.

For more details about these files and how they isolate resource types in multiple environments, see [environment isolation](#).

SSL and certificates

Network communications and security in Puppet Enterprise are based on HTTPS, which secures traffic using X.509 certificates. PE includes its own CA tools, which you can use to regenerate certs as needed.

- [Regenerate the console certificate](#) on page 678

The console certificate expires every 824 days. Regenerate the console certificate when it's nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.

- [Regenerate infrastructure certificates](#) on page 678

Regenerating certificates and security credentials—both private and public keys—created by the built-in PE certificate authority can help ensure the security of your installation in certain cases.

- [Use an independent intermediate certificate authority](#) on page 681

The built-in Puppet certificate authority automatically generates a root and intermediate certificate, but if you need additional intermediate certificates or prefer to use a public authority CA, you can set up an independent intermediate certificate authority. You must complete this configuration during installation.

- [Use a custom SSL certificate for the console](#) on page 683

The console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. You might find that this is not an acceptable scenario and want to use a custom CA to create the console's certificate.

- [Change the hostname of a primary server](#) on page 684

To change the hostnames assigned to your PE infrastructure nodes, you must update the corresponding PE certificate names. You might have to update your primary server hostname, for example, when migrating to a new PE installation, or after an organizational change such as a change in company name.

- [Generate a custom Diffie-Hellman parameter file](#) on page 685

The "Logjam Attack" (CVE-2015-4000) exposed several weaknesses in the Diffie-Hellman (DH) key exchange. To help mitigate the "Logjam Attack," PE ships with a pre-generated 2048 bit Diffie-Hellman param file. In the case that you don't want to use the default DH param file, you can generate your own.

- [Enable TLSv1](#) on page 686

TLSv1 and TLSv1.1 are disabled by default in PE.

Regenerate the console certificate

The console certificate expires every 824 days. Regenerate the console certificate when it's nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.

After you delete the existing console certificate and its artifacts, a new certificate is generated automatically on the next Puppet run.

1. On your primary server, run:

```
rm /etc/puppetlabs/puppetserver/ca/signed/console-cert.pem /etc/
puppetlabs/puppet/ssl/certs/console-cert.pem /etc/puppetlabs/puppet/ssl/
private_keys/console-cert.pem /etc/puppetlabs/puppet/ssl/public_keys/
console-cert.pem
```

2. Run `Puppet, puppet agent -t`, or wait for the next Puppet run.

Regenerate infrastructure certificates

Regenerating certificates and security credentials—both private and public keys—created by the built-in PE certificate authority can help ensure the security of your installation in certain cases.

The process for regenerating certificates varies depending on your goal.

If your goal is to...	Do this...
Upgrade to the intermediate certificate architecture introduced in Puppet 6.0.	Complete these tasks in order:
Fix a compromised or damaged certificate authority.	<ol style="list-style-type: none"> 1. Delete and recreate the certificate authority on page 679 2. Regenerate compiler certificates on page 679, if applicable 3. Regenerate agent certificates on page 680 4. Regenerate replica certificates on page 680
Fix a compromised compiler certificate or troubleshoot SSL errors on compilers.	Regenerate compiler certificates on page 679
Fix a compromised agent certificate or troubleshoot SSL errors on agent nodes.	Regenerate agent certificates on page 680
Specify a new DNS alt name or other trusted data.	Regenerate primary server certificates on page 681

Note: To support recovery, backups of your certificates are saved and the location of the backup directory is output to the console. If the command fails after deleting the certificates, you can restore your certificates with the contents of this backup directory.

Delete and recreate the certificate authority

Recreate the certificate authority only if you're upgrading to the new certificate architecture introduced in Puppet 6.0, or if your certificate authority was compromised or damaged beyond repair.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see [Bolt OpenSSH configuration options](#).



CAUTION: Replacing your certificate authority invalidates all existing certificates in your environment. Complete this task only if and when you're prepared to regenerate certificates for both your infrastructure nodes (including external PE-PostgreSQL in extra-large installations) and your entire agent fleet.

On your primary server logged in as root, run:

```
puppet infrastructure run rebuild_certificate_authority
```

The SSL and cert directories on your CA server are backed up with `"_bak"` appended to the end, CA files are removed and certificates are rebuilt, and a Puppet run completes.

Regenerate compiler certificates

Regenerate compiler certificates to fix a compromised certificate or troubleshoot SSL errors on compilers, or if you recreated your certificate authority.

On your primary server logged in as root, run the following command, specifying any additional parameters required for your environment and use case:

```
puppet infrastructure run regenerate_compiler_certificate
target=<COMPILER_HOSTNAME>
```

- **If you use DNS alternative names**, specify `dns_alt_names` as a comma-separated list of names to add to agent certificates.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- **If you recreated your certificate authority**, or are otherwise unable to connect to the compiler with the orchestrator, specify `clean_crl=true` and `--use-ssh`, as well as any additional parameters needed to connect over SSH.

The compiler's SSL directory is backed up to `/etc/puppetlabs/puppet/ssl_bak`, its certificate is regenerated and signed, a Puppet run completes, and the compiler resumes its role in your deployment.

Regenerate agent certificates

Regenerate *nix or Windows agent certificates to fix a compromised certificate or troubleshoot SSL errors on agents, or if you recreated your certificate authority.

On your primary server logged in as root, run the following command, specifying any additional parameters required for your environment and use case:

```
puppet infrastructure run regenerate_agent_certificate
agent=<AGENT_HOSTNAME_1>,<AGENT_HOSTNAME_2>
```

- **If you use DNS alternative names**, specify `dns_alt_names` as a comma-separated list of names to add to agent certificates.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- **If you recreated your certificate authority**, or are otherwise unable to connect to nodes with the orchestrator, specify `clean_crl=true` and `--use-ssh`, as well as any additional parameters needed to connect over SSH.

Agent SSL directories are backed up to `/etc/puppetlabs/puppet/ssl_bak` (*nix) or `C:\ProgramData\PuppetLabs/puppet/etc/ssl_bak` (Windows), their certificates are regenerated and signed, a Puppet run completes, and the agents resume their role in your deployment.

Regenerate replica certificates

Regenerate replica certificates for your disaster recovery installation to specify a new DNS alt name or other trusted data, or if you recreated your certificate authority.

On your primary server logged in as root, run the following command, specifying any additional parameters required for your environment and use case:

```
puppet infrastructure run regenerate_replica_certificate
target=<REPLICA_HOSTNAME>
```

- **If you use DNS alternative names**, specify `dns_alt_names` as a comma-separated list of names to add to agent certificates.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- **If you recreated your certificate authority**, or are otherwise unable to connect to the replica with the orchestrator, specify `clean_crl=true` and `--use-ssh`, as well as any additional parameters needed to connect over SSH.

The replica's SSL directory is backed up to `/etc/puppetlabs/puppet/ssl_bak`, its certificate is regenerated and signed, a Puppet run completes, and the replica resumes its role in your deployment.

Regenerate primary server certificates

Regenerate primary server certificates to specify a new DNS alt name or other trusted data. This process regenerates the certificates for all primary infrastructure nodes, including external PE-PostgreSQL in extra-large installations.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see [Bolt OpenSSH configuration options](#).

To view all available parameters, use the `--help` flag. The logs for all `puppet infrastructure run` Bolt plans are located at `/var/log/puppetlabs/installer/bolt_info.log`.

On your primary server logged in as root, run:

```
puppet infrastructure run regenerate_master_certificate
```

You can specify these optional parameters:

- `dns_alt_names` – Comma-separated list of alternate DNS names to be added to the certificates generated for your primary server.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your primary server certificate.

- `tmpdir` — Path to a directory to use for uploading and executing temporary files.

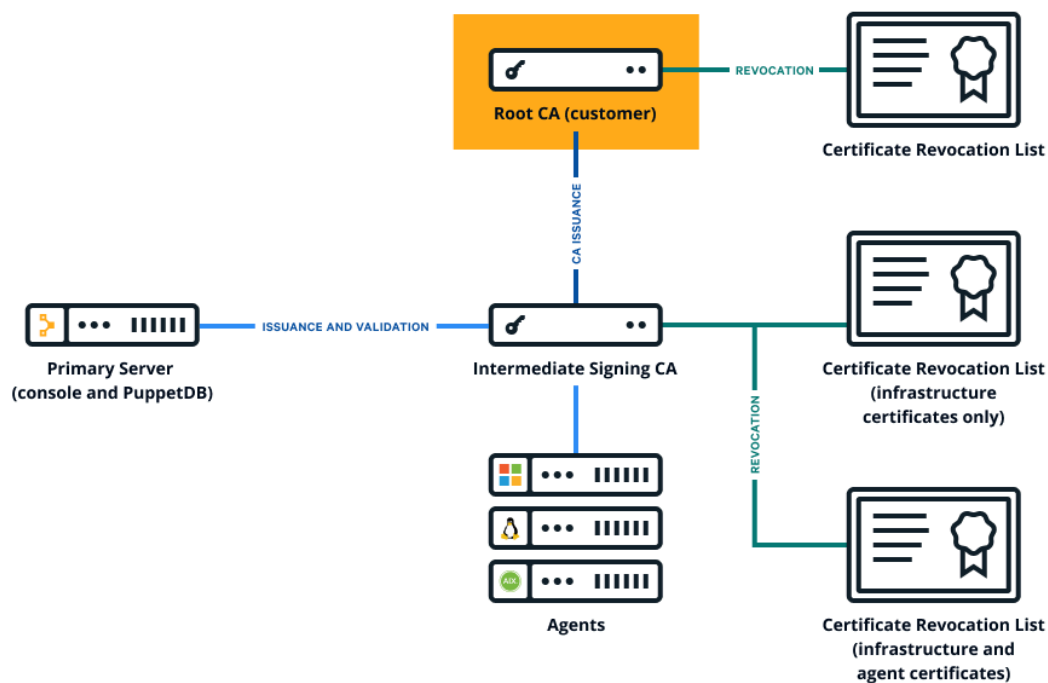
Use an independent intermediate certificate authority

The built-in Puppet certificate authority automatically generates a root and intermediate certificate, but if you need additional intermediate certificates or prefer to use a public authority CA, you can set up an independent intermediate certificate authority. You must complete this configuration during installation.



CAUTION: This method requires more manual maintenance than the default certificate authority setup. With an external chain of trust, you must monitor for and promptly update expired CRLs, because an expired CRL anywhere in the chain causes certificate validation failures. To manage an external CRL chain:

- Take note of the `Next Update` dates of the CRLs for your entire chain of trust.
- Submit updated CRLs to Puppet Server.
- Configure agents to download CRL updates by setting `crl_refresh_interval` in the `puppet_enterprise::profile::agent` class.



1. Collect the PEM-encoded certificates and CRLs for your organization's chain of trust, including the root certificate, any intermediate certificates, and the signing certificate. (The signing certificate might be the root or intermediate certificate.)
2. Create a private RSA key, with no passphrase, for the Puppet CA.
3. Create a PEM-encoded Puppet CA certificate.
 - a) Create a CSR for the Puppet CA.
 - b) Generate the Puppet CA certificate by signing the CSR using your external CA.

Ensure the CA constraint is set to true and the `keyIdentifier` is composed of the 160-bit SHA-1 hash of the value of the bit string `subjectPublicKeyField`. See RFC 5280 section 4.2.1.2 for details.
4. Concatenate all of the certificates into a PEM-encoded certificate bundle, starting with the Puppet CA cert and ending with your root certificate.

```
-----BEGIN CERTIFICATE-----
<PUPPET CA CERTIFICATE>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<OPTIONAL INTERMEDIATE CA CERTIFICATES>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<ROOT CA CERTIFICATE>
-----END CERTIFICATE-----
```

- Concatenate all of the CRLs into a PEM-encoded CRL chain, starting with any optional intermediate CA CRLs and ending with your root certificate CRL.

```
-----BEGIN X509 CRL-----
<OPTIONAL INTERMEDIATE CA CRLs>
-----END X509 CRL-----
-----BEGIN X509 CRL-----
<ROOT CA CRL>
-----END X509 CRL-----
```

Tip: The PE installer automatically generates a Puppet CRL during installation, so you don't have to manage the Puppet CRL manually.

- Copy your CA bundle (from step 4 on page 682), CRL chain (from step 5 on page 683), and private key (from step 2 on page 682) to the node where you're installing the primary server.

Tip: Allow access to your private key only from the PE installation process, which runs as root.

- Install PE using a customized `pe.conf` file with `signing_ca` parameters: `./puppet-enterprise-installer -c <PATH_TO_pe.conf>`

In your customized `pe.conf` file, you must include three keys for the `signing_ca` parameter: `bundle`, `crl_chain`, and `private_key`.

```
{
  "pe_install::signing_ca": {
    "bundle": "/root/ca/int_ca_bundle"
    "crl_chain": "/root/ca/int_crl_chain"
    "private_key": "/root/ca/int_key"
  }
}
```

- Optional: Validate that the CA is working by running `puppet agent -t` and verifying your intermediate CA with OpenSSL.

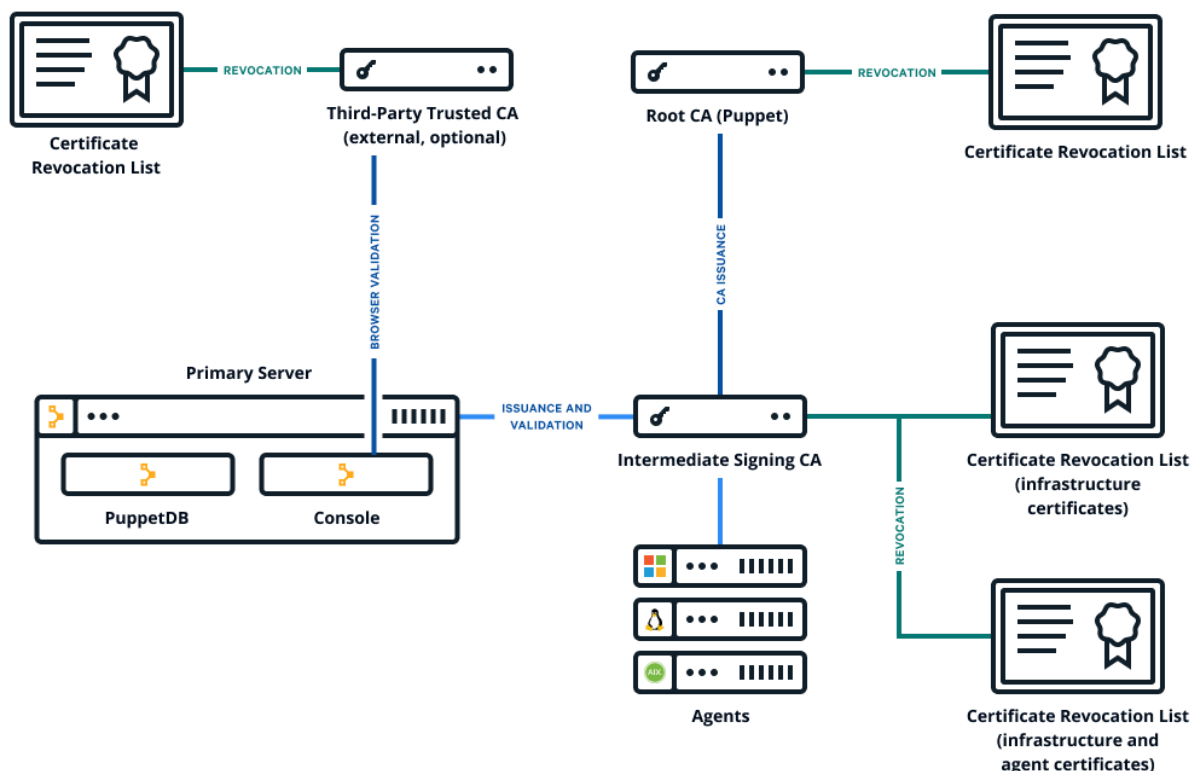
```
openssl x509 -in /etc/puppetlabs/puppet/ssl/ca/signed/<HOSTNAME>.crt
-text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=intermediate-ca
  ...
```

Use a custom SSL certificate for the console

The console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. You might find that this is not an acceptable scenario and want to use a custom CA to create the console's certificate.

Before you begin

- You should have a X.509 cert, signed by the custom party CA, in PEM format, with matching private and public keys.
- If your custom cert is issued by an intermediate CA, the CA bundle needs to contain a complete chain, including the applicable root CA.
- The keys and certs used in this procedure must be in PEM format.



1. Retrieve the custom certificate and private key.
2. Move the certificate to `/etc/puppetlabs/puppet/ssl/certs/console-cert.pem`, replacing any existing file named `console-cert.pem`.
3. Move the private key to `/etc/puppetlabs/puppet/ssl/private_keys/console-cert.pem`, replacing any existing file named `console-cert.pem`.
4. If you previously specified a custom SSL certificate, remove any `browser_ssl_cert` and `browser_ssl_private_key` parameters.
 - a) In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Console** group.
 - b) On the **Configuration data** tab, in the `puppet_enterprise::profile::console` class, remove any values for `browser_ssl_cert` and `browser_ssl_private_key` and commit changes.
5. Run Puppet: `puppet agent -t`

You can navigate to your console and see the custom certificate in your browser.

Change the hostname of a primary server

To change the hostnames assigned to your PE infrastructure nodes, you must update the corresponding PE certificate names. You might have to update your primary server hostname, for example, when migrating to a new PE installation, or after an organizational change such as a change in company name.

Before you begin

Download and install the [puppetlabs-support_tasks](#) module.

Make sure you are using the latest version of the `support_tasks` module.

Set up RBAC permissions and token authentication for orchestrator tasks. See [Setting PE RBAC permissions and token authentication for orchestrator](#) on page 473 for more information.

To update the hostname of your primary server:

1. On the primary server, set the new hostname by running the following command:

```
hostnamectl set-hostname newhostname.example.com
```

2. Make sure the `hostname -f` command returns the new fully qualified hostname, and that it resolves to the same IP address as the old hostname, by adding an entry for the new hostname in `/etc/hosts`:

```
<IP address> <newhostname.example.com> <oldhostname.example.com> ...
```

3. Run a task for changing the host name against the old certificate name on the primary server, using one of the following methods:

- Using Bolt:

```
bolt task run support_tasks::st0263_rename_pe_master -n $(puppet config
  print certname) --modulepath="/etc/puppetlabs/code/environments/
  production/modules"
```

Note: When running the task, Bolt must be using the default SSH transport, rather than the PCP protocol, to avoid errors when services are restarted.

- Using the command line:

```
puppet task run support_tasks::st0263_rename_pe_master -n $(puppet
  config print certname)
```

The task restarts all Puppet services, which causes a connection error. You can ignore the error while the task continues to run in the background. To check if the task is complete, tail `/var/log/messages`. When the output from the `puppet agent -t` command is displayed in the system log, the task is complete. For example:

```
# tail /var/log/messages
Aug 15 09:08:28 oldhostname systemd: Reloading pe-orchestration-services
  Service.
Aug 15 09:08:29 oldhostname systemd: Reloaded pe-orchestration-services
  Service.
Aug 15 09:08:29 oldhostname puppet-agent[4780]: (/
  Stage[main]/Puppet_enterprise::Profile::Orchestrator/
  Puppet_enterprise::Trapperkeeper::Pe_service[orchestration-services]/
  Service[pe-orchestration-services]) Triggered 'refresh' from 1 event
Aug 15 09:08:34 oldhostname puppet-agent[4780]: Applied catalog in 19.16
  seconds
```

Generate a custom Diffie-Hellman parameter file

The "Logjam Attack" (CVE-2015-4000) exposed several weaknesses in the Diffie-Hellman (DH) key exchange. To help mitigate the "Logjam Attack," PE ships with a pre-generated 2048 bit Diffie-Hellman param file. In the case that you don't want to use the default DH param file, you can generate your own.

Note: In the following procedure, `<PROXY-CUSTOM-dhparam>.pem` can be replaced with any file name, except `dhparam_puppetproxy.pem`, as this is the default file name used by PE.

1. On your primary server, run:

```
/opt/puppetlabs/puppet/bin/openssl dhparam -out /etc/puppetlabs/nginx/
<PROXY-CUSTOM-dhparam>.pem 2048
```

Note: After running this command, PE can take several minutes to complete this step.

2. Open your `pe.conf` file (located at `/etc/puppetlabs/enterprise/conf.d/pe.conf`) and add the following parameter and value:

```
"puppet_enterprise::profile::console::proxy::dhparam_file": "/etc/
puppetlabs/nginx/<PROXY-CUSTOM-dhparam>.pem"
```

3. Run Puppet: `puppet agent -t`.

Enable TLSv1

TLSv1 and TLSv1.1 are disabled by default in PE.

You must enable TLSv1 to install agents on these platforms:

- AIX
- Solaris 11

To comply with security regulations, PE 2019.1 and later uses only version 1.2 of the Transport Layer Security (TLS) protocol.

1. In the console, click **Node groups** > **PE Infrastructure**.
2. On the **Configuration data** tab, add the following class, parameter, and value:

Class	Parameter	Value
<code>puppet_enterprise::master::puppetserver</code>	<code>ssl::puppetserver</code>	<code>["TLSv1", "TLSv1.1", "TLSv1.2"]</code>

3. Click **Add data**, and commit changes.
4. Run Puppet on the primary server and any compilers.

Maintenance

- [Backing up and restoring Puppet Enterprise](#) on page 686

Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new primary server, troubleshoot, and quickly recover in the case of system failures.

- [Database maintenance](#) on page 693

You can optimize the Puppet Enterprise (PE) databases to improve performance. For database maintenance, we recommend using the [pe_databases](#) module.

Backing up and restoring Puppet Enterprise

Keep regular backups of your PE infrastructure. Backups allow you to more easily migrate to a new primary server, troubleshoot, and quickly recover in the case of system failures.

Note: These instructions are for standard installations only.

Important: Always store your backups in a safe location that you can access if your primary server fails. Backup files are not encrypted, so secure these as you would any sensitive information.

You can use the `puppet-backup` command to back up and restore your primary server. You can't use this command to back up compilers. By default, the backup command creates a backup of:

- Your PE configuration, including license, classification, and RBAC settings. However, configuration backup does not include Puppet gems or Puppet Server gems.
- PE CA certificates and the full SSL directory.
- The Puppet code deployed to your code directory at backup time.
- PuppetDB data, including facts, catalogs and historical reports.

Each time you create a new backup, PE creates a single, timestamped backup file, named in the format `pe_backup-<TIMESTAMP>.tgz`. This file includes everything you're backing up. By default, PE writes backup files to `/var/puppetlabs/backups`, but you can change this location when you run the backup command. When you restore, specify the backup file you want to restore from.

If you are restoring to a previously existing primary server, uninstall and reinstall PE before restoring your infrastructure. If you are restoring or migrating your infrastructure to a primary server with a different hostname than the previous primary server, you'll redirect your agents to the new primary server during the restore process. In both cases, the freshly installed PE must be the same PE version that was in use when you backed up the files.

By default, backup and restore functions include your Puppet configuration, certificates, code, and PuppetDB. However, you can limit the scope of backup and restore with command line options. This allows you to back up to or restore from multiple files. This is useful if you want to back up some parts of your infrastructure more often than others.

For example, if you have frequent code changes, you might back up the code more often than you back up the rest of your infrastructure. When you limit backup scope, the backup file contains only the specified parts of your infrastructure. Be sure to give your backup file a name that identifies the scope so that you always know what a given file contains.

During restore, you must restore all scopes: code, configuration, certificates, and PuppetDB. However, you can restore each scope from different files, either by restoring from backup files with limited scope or by limiting the scope of the restore. For example, by specifying scope when you run the restore command, you could restore code, configuration, and certificates from one backup file and PuppetDB from a different one.

Back up your infrastructure

PE backup creates a copy of your primary server, including configuration, certificates, code, and PuppetDB.

By default, PE backs up files to `/var/puppetlabs/backups` and names them with a timestamp. See the backup and restore [reference](#) for a complete list of options for customizing your backup.

1. On your primary server logged in as root, run `puppet-backup create --dir=<BACKUP_DIRECTORY>`, specifying a separate disk as your backup directory.

Backup can take up to several hours depending on the size of PuppetDB.

2. Back up the inventory service secret key located at `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`

Secure the key as you would any sensitive information.



CAUTION: The backup command does not include the secret key. You must back up this information separately.

Related information

[Configuration parameters and the `pe.conf` file](#) on page 103

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

Restore your infrastructure

Use the restore commands to migrate your primary server to a new host or to recover from system failure.

Before you begin

Remember that you must restore files to a fresh installation of the same version of PE used in your backup file.

1. If you are restoring to an existing primary server, purge any existing PE installation from it.
 - a) On your primary server, uninstall PE by running `sudo /opt/puppetlabs/bin/puppet-enterprise-uninstaller -p -d`
 - b) Ensure that the directories `/opt/puppetlabs/` and `/etc/puppetlabs/` are no longer present on the system.

For details about uninstalling PE, see the [uninstalling](#) documentation.

2. Install PE on the primary server you are restoring to. This must be the same PE version that was used for the backup files.
 - a) If you don't have the PE installer script on the machine you want to restore to, download the installer tarball to the machine from the [download](#) site, and unpack it by running `tar -xf <TARBALL_FILENAME>`
 - b) Navigate to the directory containing the install script. The installer script is located in the PE directory created when you unpacked the tarball.
 - c) Install PE by running `sudo ./puppet-enterprise-installer`

For details about the PE installation process, see the [installing](#) documentation.

3. On the primary server logged in as root, restore your PE infrastructure by running `puppet-backup restore <BACKUP-FILENAME>`

To change the default command behavior, pass option flags and values to the command. See the [command reference](#) for a complete list of options.

For example, to restore your scope, certificates, and code from one backup file, but your PuppetDB data from another, pass the `--scope` option:

```
puppet-backup restore pe_backup-2018-03-03_20.07.15.UTC.tgz --
scope=config,certs,code

puppet-backup restore pe_backup-2018-04-04_20.07.15.UTC.tgz --
scope=puppetdb
```

4. Restore your backup of the secret key used to encrypt and decrypt sensitive data stored in the inventory service. The secret key is stored at: `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`



CAUTION: The restore command does not include the secret key. You must restore this information separately.

5. Ensure the secret key ownership is `pe-orchestration-services:pe-orchestration-services`.
6. Restart the `pe-orchestration-services` service.

7. If you restored PE onto a primary server with a different hostname than the original installation, and you have not configured the `dns_alt_names` setting in the `pe.conf` file, redirect your agents to the new primary server. An easy way to do this is by running a task with the Bolt task runner.

- Download and [install Bolt](#), if you don't already have it installed.
- Update the `puppet.conf` file to point all agents at the new primary server by running:

```
bolt task run puppet_conf action=set section=agent setting=server
value=<RESTORE_HOSTNAME> --nodes <COMMA-SEPARATED LIST OF NODES>
bolt task run puppet_conf action=set section=main setting=server
value=<RESTORE_HOSTNAME> --nodes <COMMA-SEPARATED LIST OF NODES>
```

- Run `puppet agent -t --no-use_cached_catalog` on the newly restored primary server **twice** to apply changes and then restart services.
 - Run `puppet agent -t --no-use_cached_catalog` on all agent nodes to test connection to the new primary server.
8. If your infrastructure had Code Manager enabled when your backup file was created, deploy your code by running:

```
puppet-access login
puppet code deploy --all --wait
```

Related information

[Configuration parameters and the `pe.conf` file](#) on page 103

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure PE. A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

[Installing Puppet Enterprise](#) on page 102

Installing PE begins with setting up a standard installation. From here, you can scale up to the large or extra-large installation as your infrastructure grows, or customize configuration as needed.

[Uninstalling](#) on page 142

Puppet Enterprise includes a script for uninstalling. You can uninstall component infrastructure nodes or from agent nodes.

Backup and restore reference

Use these options to change the backup and restore scope and other options for the `puppet-backup` command.

Note: Backup commands must be run as root.

`puppet-backup create`

Run the `puppet-backup create` command to create backup files of your PE infrastructure.

Usage:

```
puppet-backup create [--dir=<DIRECTORY_PATH>] [--name=<BACKUP_NAME>.tgz] [--
scope=<SCOPE_LIST>] [--force]
```

Option	Description	Values	Default
<code>--dir=BACKUP_DIR</code>	Specifies the directory to write the backup file to.	A valid filepath that the <code>pe-postgres</code> user has write permission for.	<code>/var/puppetlabs/backups/</code>
<code>--name=BACKUP_NAME.tgz</code>	Specifies the name for the backup file.	A string designating a file name.	<code>pe_backup-<TIMESTAMP>.tgz</code>

Option	Description	Values	Default
<code>--pe-environment=ENVIRONMENT</code>	Specifies the environment to back up. To ensure configuration is recovered correctly, this must be the environment where your primary server is located.	A valid environment name.	production
<code>--scope=SCOPE</code>	Scope of backup to create.	Either all or any combination of the other available scopes, as a comma-separated list: <ul style="list-style-type: none"> • <code>config</code>: PE configuration including license, classification, and RBAC settings. Does not include Puppet gems or Puppet Server gems. • <code>certs</code>: PE CA certificates and full SSL directory • <code>code</code>: Puppet code deployed to your codedir at backup time. • <code>puppetdb</code>: PuppetDB data, including facts, catalogs, and historical reports • <code>all</code>: All listed scopes. 	all
<code>--force</code>	Bypass validation checks and ignore warnings.	None.	If you don't specify <code>--force</code> , PE verifies that the destination directory exists and has enough space for the backup process.

For example, to create a backup of PuppetDB only and give it a name that shows the scope of the file:

```
puppet-backup create --scope=puppetdb --name=puppetdb_backup_03032018.tgz
```

puppet-backup restore

Run the `puppet-backup restore` command to restore your PE infrastructure from backup files.

Usage:

```
puppet-backup restore <PATH/TO/BACKUP_FILE.tgz> [--scope=<SCOPE_LIST>] [--force]
```

Option	Description	Values	Default
<code>--pe-environment=ENVIRONMENT</code>	Specifies the environment to restore.	A valid environment name for which you have an existing backup.	production
<code>--scope=SCOPE</code>	Scope of backup to restore. All scopes must eventually be restored, but you can restore different scopes from different backup files with successive restore commands.	Either all or any combination of the other available scopes, as a comma-separated list: <ul style="list-style-type: none"> <code>config</code>: PE configuration including license, classification, and RBAC settings. Does not include Puppet gems or Puppet Server gems. <code>certs</code>: PE CA certificates and full SSL directory <code>code</code>: Puppet code deployed to your codedir at backup time. <code>puppetdb</code>: PuppetDB data, including facts, catalogs, and historical reports <code>all</code>: All listed scopes. 	all

Option	Description	Values	Default
<code>--force</code>	Bypass validation checks and ignore warnings.	None.	If you don't specify <code>--force</code> , PE verifies that the destination directory exists and has enough space for the restore process. Returns warnings for insufficient space or invalid locations.

For example, to restore PuppetDB from one backup file and restore configuration, certificates, and code from another backup file:

```
puppet-backup restore /mybackups/pe_backup_03032018.tgz --scope=puppetdb
puppet-backup restore /mybackups/pe_backup_04042018.tgz --
scope=config,certs,code
```

Directories and data backed up

Scope	Directories and databases backed up
certs	<ul style="list-style-type: none"> <code>/etc/puppetlabs/puppet/ssl/</code>
code	<ul style="list-style-type: none"> <code>/etc/puppetlabs/code/</code> <code>/etc/puppetlabs/code-staging/</code> <code>/opt/puppetlabs/server/data/puppetserver/filesync/storage/</code>

Scope	Directories and databases backed up
config	<ul style="list-style-type: none"> • Orchestrator database • RBAC database • Classifier database • /etc/puppetlabs/ , except: <ul style="list-style-type: none"> • /etc/puppetlabs/code/ • /etc/puppetlabs/code-staging/ • /etc/puppetlabs/puppet/ssl • /opt/puppetlabs/ , except: <ul style="list-style-type: none"> • /opt/puppetlabs/puppet • /opt/puppetlabs/server/pe_build • /opt/puppetlabs/server/data/packages • /opt/puppetlabs/server/apps • /opt/puppetlabs/server/data/postgresql • /opt/puppetlabs/server/data/enterprise/modules • /opt/puppetlabs/server/data/puppetserver/vendored-jruby-gems • /opt/puppetlabs/bin • /opt/puppetlabs/client-tools • /opt/puppetlabs/server/share • /opt/puppetlabs/server/data/puppetserver/filesync/storage • /opt/puppetlabs/server/data/puppetserver/filesync/client
puppetdb	<ul style="list-style-type: none"> • PuppetDB • /opt/puppetlabs/server/data/puppetdb

Database maintenance

You can optimize the Puppet Enterprise (PE) databases to improve performance. For database maintenance, we recommend using the [pe_databases](#) module.

Databases in Puppet Enterprise

PE uses PostgreSQL as the backend for its databases. Use the native tools in PostgreSQL to perform database exports and imports.

The PE PostgreSQL database includes the following databases:

Database	Description
pe-activity	Activity data from the Classifier, including user, nodes, and time of activity.
pe-classifier	Classification data, all node group information.
pe-puppetdb	PuppetDB data, including exported resources, catalogs, facts, and reports.
pe-rbac	Role-based access control data, including users, permissions, and AD/LDAP info.
pe-orchestrator	Orchestrator data, including user, node, and result details about job runs.

List all database names

Use these instructions to list all database names.

To generate a list of database names:

1. Assume the `pe-postgres` user:

```
sudo su - pe-postgres -s /bin/bash
```

2. Open the PostgreSQL command-line:

```
/opt/puppetlabs/server/bin/psql
```

3. List the databases:

```
\l
```

4. Exit the PostgreSQL command line:

```
\q
```

5. Log out of the `pe-postgres` user:

```
logout
```

Troubleshooting

Use this guide to troubleshoot issues with your Puppet Enterprise installation.

Important: Before you troubleshoot, view the [What gets installed and where?](#) on page 97 page. PE installs several software components, configuration files, databases, services and users, and log files. It is useful to know their locations when you need to troubleshoot your infrastructure.

- [Log locations](#) on page 695

The software distributed with PE generates log files you can use for troubleshooting.

- [Troubleshooting installation](#) on page 696

If installation fails, check for these issues.

- [Troubleshooting disaster recovery](#) on page 697

If disaster recovery commands fail, check for these issues.

- [Troubleshooting puppet infrastructure run commands](#) on page 697

If `puppet infrastructure run` commands fail, review the logs at `/var/log/puppetlabs/installer/bolt_info.log` and check for these issues.

- [Troubleshooting connections between components](#) on page 698

If agent nodes can't retrieve configurations, check for communication, certificate, DNS , and NTP issues.

- [Troubleshooting the databases](#) on page 700

If you have issues with the databases that support the console, make sure that the PostgreSQL database is not too large or using too much memory, that you don't have port conflicts, and that `puppet apply` is configured correctly.

- [Troubleshooting backup and restore](#) on page 701

If backup or restore fails, check for these issues.

- [Troubleshooting Code Manager](#)

- [Troubleshooting Windows](#) on page 701

Troubleshoot Windows issues with failed installations and upgrades, and failed or incorrectly applied manifests. Use the error message reference to solve your issues. Enable debugging.

Log locations

The software distributed with PE generates log files you can use for troubleshooting.

Primary server logs

- `/var/log/puppetlabs/puppetserver/code-manager-access.log`
- `/var/log/puppetlabs/puppetserver/file-sync-access.log`
- `/var/log/puppetlabs/puppetserver/masterhttp.log`
- `/var/log/puppetlabs/puppetserver/pcp-broker.log` — The log file for Puppet Communications Protocol brokers on compilers.
- `/var/log/puppetlabs/puppetserver/puppetserver.log` — The primary server logs its activity, including compilation errors and deprecation warnings, here.
- `/var/log/puppetlabs/puppetserver/puppetserver-access.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/puppetserver/puppetserver-status.log`

Agent logs

The locations of agent logs depend on the agent operating system.

On *nix nodes, the agent service logs its activity to the syslog service. Your syslog configuration dictates where these messages are saved, but the default location is `/var/log/messages` on Linux, `/var/log/system.log` on macOS, and `/var/adm/messages` on Solaris.

On Windows nodes, the agent service logs its activity to the event log. You can view its logs by browsing the event viewer.

Console and console services logs

- `/var/log/puppetlabs/console-services/console-services.log`
- `/var/log/puppetlabs/console-services/console-services-api-access.log`

- `/var/log/puppetlabs/console-services-access.log`
- `/var/log/puppetlabs/console-services-daemon.log` — This is where fatal errors or crash reports can be found.
- `/var/log/puppetlabs/nginx/access.log`
- `/var/log/puppetlabs/nginx/error.log` — Contains errors related to nginx. Console errors that aren't logged elsewhere can be found in this log.

Installer logs

- `/var/log/puppetlabs/installer/http.log` — Contains web requests sent to the installer. This log is present only on the machine from which a web-based installation was performed.
- `/var/log/puppetlabs/installer/orchestrator_info.log` — Contains run details about puppet infrastructure commands that use the orchestrator, including the commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.
- `/var/log/puppetlabs/installer/install_log.lastrun.<hostname>.log` — Contains the contents of the last installer run.
- `/var/log/puppetlabs/installer/installer-<timestamp>.log` — Contains the operations performed and any errors that occurred during installation.
- `/var/log/puppetlabs/installer/<action_name>_<timestamp>_<run_description>.log` — Contains details about disaster recovery command execution. Each action triggers multiple Puppet runs, some on the primary server, some on the replica, so there might be multiple log files for each command.

Database logs

- `/var/log/puppetlabs/postgresql/9.6/pgstartup.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Mon.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Tue.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Wed.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Thu.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Fri.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Sat.log`
- `/var/log/puppetlabs/postgresql/9.6/postgresql-Sun.log`
- `/var/log/puppetlabs/puppetdb/puppetdb.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-access.log`
- `/var/log/puppetlabs/puppetdb/puppetdb-status.log`

Troubleshooting installation

If installation fails, check for these issues.

Note: If you encounter errors during installation, you can troubleshoot and run the installer as many times as needed.

DNS is misconfigured

DNS must be configured correctly for successful installation.

1. Verify that agents can reach the primary server hostname you chose during installation.
2. Verify that the primary server can reach *itself* at the primary server hostname you chose during installation.
3. If the primary server and console components are on different servers, verify that they can communicate with each other.

Security settings are misconfigured

Firewall and security settings must be configured correctly for successful installation.

1. On your primary server, verify that inbound traffic is allowed on ports 8140 and 443.
2. If your primary server has multiple network interfaces, verify that the primary server allows traffic via the IP address that its valid DNS names resolve to, not just via an internal interface.

Troubleshooting disaster recovery

If disaster recovery commands fail, check for these issues.

Latency over WAN

If the primary server and replica communicate over a slow, high latency, or lossy connection, the provision and enable commands can fail.

If this happens, try re-running the command.

Replica is connected to a compiler instead of a primary server

The provision command triggers an error if you try to provision a replica node that's connected to a compiler. The error is similar to the following:

```
Failure during provision command during the puppet agent run on replica 2:
Failed to generate additional resources using 'eval_generate':
  Error 500 on SERVER: Server Error: Not authorized
  to call search on /file_metadata/pe_modules with
  {:rest=>"pe_modules", :links=>"manage", :recurse=>true, :source_permissions=>"ignore",
Source: /Stage[main]/Puppet_enterprise::Profile::Primary_master_replica/
File[/opt/puppetlabs/server/share/installer/modules]File: /opt/
puppetlabs/puppet/modules/puppet_enterprise/manifests/profile/
primary_master_replica.ppLine: 64
```

On the replica you want to provision, edit `/etc/puppetlabs/puppet.conf` so that the `server` and `server_list` settings use a primary server, rather than a compiler.

Both `server` and `server_list` are set in the agent configuration file

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

Node groups are empty

When provisioning and enabling a replica, the orchestrator is used to run Puppet on different groups of nodes. If a group of nodes is empty, the tool reports that there's nothing for it to do and the job is marked as failed in the output of `puppet job show`. This is expected, and doesn't indicate a problem.

Troubleshooting puppet infrastructure run commands

If `puppet infrastructure run` commands fail, review the logs at `/var/log/puppetlabs/installer/bolt_info.log` and check for these issues.

Running commands when logged in as a non-root user

All `puppet infrastructure run` commands require you to act as the root user on all nodes that the command touches. If you are trying to run a `puppet infrastructure run` command as a non-root user, you must be able to SSH to the impacted nodes (as the same non-root user) in order for the command to succeed.

When you run a `puppet infrastructure run` command, Bolt uses your system's existing OpenSSH `ssh_config` configuration file to connect to your nodes. If this file is missing or configured incorrectly, Bolt tries to connect as root. To make sure the correct user connects to the nodes, you have the following options.

1. Set up your OpenSSH `ssh_config` configuration file to point to a user with sudo privileges:

```
Host *.example.net
  UserKnownHostsFile=~/.ssh/known_hosts
  User <USER WITH SUDO PRIVILEGES>
```

2. When running a `puppet infrastructure run` command, pass in the `--user <USER WITH SUDO PRIVILEGES>` flag.

If your sudo configuration requires a password to run commands, pass in the `--sudo-password <PASSWORD>` flag when running a `puppet infrastructure run` command.

Note: To avoid logging your password to `.bash_history`, set `HISTCONTROL=ignorespace` in your `.bashrc` file and add a space to the beginning of the command.

If your operating system distribution includes the `requiretty` option in its `/etc/sudoers` file, you must either remove this option from the file or pass the `--tty` flag when running a `puppet infrastructure run` command.

Passing hashes from the command line

When passing a hash on the command line as part of a `puppet infrastructure run` command, the hash must be quoted, much like a JSON object. For example:

```
'{"parameter_one": "value_one", "parameter_two": "value_two"}'
```

Troubleshooting connections between components

If agent nodes can't retrieve configurations, check for communication, certificate, DNS, and NTP issues.

Agents can't reach the primary server

Agent nodes must be able to communicate with the primary server in order to retrieve configurations.

If agents can't reach the primary server, running `telnet <PRIMARY_HOSTNAME> 8140` returns the error "Name or service not known."

1. Verify that the primary server is reachable at a DNS name your agents recognize.
2. Verify that the `pe-puppetserver` service is running.

Agents don't have signed certificates

Agent certificates must be signed by the primary server.

If the node's Puppet agent logs have a warning about unverified peer certificates in the current SSL session, the agent has submitted a certificate signing request that hasn't yet been signed.

1. On the primary server, view a list of pending certificate requests: `puppet cert list`
2. Sign a specified node's certificate: `puppetserver ca sign <NODE NAME>`

Agents aren't using the primary server's valid DNS name

Agents trust the primary server only if they contact it at one of the valid hostnames specified when the primary server was installed.

On the node, if the results of `puppet agent --configprint server` don't return one of the valid DNS names you chose during installation of the primary server, the node and primary server can't establish communication.

1. To edit the primary server's hostname on nodes, in `/etc/puppetlabs/puppet/puppet.conf`, change the `server` setting to a valid DNS name.
2. To reset the primary server's valid DNS names, run:

```
/etc/init.d/pe-nginx stop
puppet cert clean <PRIMARY_CERTNAME>
puppet cert generate <PRIMARY_CERTNAME> --dns_alt_names=<COMMA-
SEPARATED_LIST_OF_DNS_NAMES>
/etc/init.d/pe-nginx start
```

Time is out of sync

The date and time must be in sync on the primary server's and agent nodes.

If time is out of sync on nodes, running `date` returns incorrect or inconsistent dates.

Get time in sync by setting up NTP. Keep in mind that NTP can behave unreliably on virtual machines.

Node certificates have invalid dates

The date and time must be in sync when certificates are created.

If certificates were signed out of sync, running `openssl x509 -text -noout -in $(puppet config print --section master ssl_dir)/certs/<NODE NAME>.pem` returns invalid dates, such as certificates dated in the future.

1. On the primary server, delete certificates with invalid dates: `puppet cert clean <NODE NAME>`
2. On nodes with invalid certificates, delete the SSL directory: `rm -r $(puppet config print --section master ssl_dir)`
3. On agent nodes, generate a new certificate request: `puppet agent --test`
4. On the primary server, sign the request: `puppetserver ca sign <NODE NAME>`

A node is re-using a certname

If a node re-uses an old node's certname and the primary server retains the previous node's certificate, the new node is unable to request a new certificate.

1. On the primary server, clear the node's certificate: `puppetserver ca clean <NODE NAME>`
2. On agent node, generate a new certificate request: `puppet agent --test`
3. On the primary server, sign the request: `puppetserver ca sign <NODE NAME>`

Agents can't reach the filebucket server

If the primary server is installed with a certname that doesn't match its hostname, agents can't back up files to the filebucket on the primary server.

If agents log errors like "could not back up," nodes are likely attempting to back up files to the wrong hostname.

On the primary server, edit `/etc/puppetlabs/code/environments/production/manifests/site.pp` so that filebucket server attribute points to the correct hostname:

```
# Define filebucket 'main':
filebucket { 'main':
  server => '<PRIMARY_DNS_NAME>',
  path   => false,
}
```

Changing the filebucket server attribute on the primary server fixes the error on all agent nodes.

Orchestrator can't connect to PE Bolt server

Debug a faulty connection between the orchestrator and PE Bolt server by setting the `bolt_server_loglevel` in the `puppet_enterprise::profile::bolt_server` class and running Puppet, or by manually updating `loglevel` in `/etc/puppetlabs/bolt-server/conf.d/bolt-server.conf`. The server logs are located at `/var/log/puppetlabs/bolt-server/bolt-server.log`.

Troubleshooting the databases

If you have issues with the databases that support the console, make sure that the PostgreSQL database is not too large or using too much memory, that you don't have port conflicts, and that `puppet apply` is configured correctly.

Note: If you're using your own instance of PostgreSQL for the console and PuppetDB, you must use version 9.1 or higher.

PostgreSQL is taking up too much space

The PostgreSQL `autovacuum=on` setting prevents the database from growing too large and unwieldy. Routine vacuuming is turned on by default.

Verify that `autovacuum` is set to `on`.

PostgreSQL buffer memory causes installation to fail

When installing PE on machines with large amounts of RAM, the PostgreSQL database might use more shared buffer memory than is available.

If this issue is present, `/var/log/pe-postgresql/pgstartup.log` shows the error:

```
FATAL: could not create shared memory segment: No space left on device
DETAIL: Failed system call was shmget(key=5432001, size=34427584512,03600).
```

1. On the primary server, set the `shmmax` kernel setting to approximately 50% of the total RAM.
2. Set the `shmall` kernel setting to the quotient of the new `shmmax` setting divided by the page size. You can confirm page size by running `getconf PAGE_SIZE`.
3. Set the new kernel settings:

```
sysctl -w kernel.shmmax=<your shmmax calculation>
sysctl -w kernel.shmall=<your shmall calculation>
```

The default port for PuppetDB conflicts with another service

By default, PuppetDB communicates over port 8081. In some cases, this might conflict with existing services, for example McAfee ePolicy Orchestrator.

Install in text mode with a parameter in `pe.conf` that specifies a different port using `puppet_enterprise::puppetdb_port`.

`puppet resource` generates Ruby errors after connecting `puppet apply` to PuppetDB

If Puppet `apply` is configured incorrectly, for example by modifying `puppet.conf` to add the parameters `storeconfigs_backend = puppetdb` and `storeconfigs = true` in both the main and primary server sections, `puppet resource` ceases to function and displays a Ruby run error.

Modify `/etc/puppetlabs/puppet/routes.yaml` to correctly [connect Puppet apply](#) without affecting other functions.

Troubleshooting backup and restore

If backup or restore fails, check for these issues.

The `puppet-backup create` command fails with the error `command puppet infrastructure recover_configuration failed`

The `puppet-backup create` command might fail if any gem installed on the Puppet Server isn't present on the agent environment on the primary server. If the gem is either absent or of a different version on the primary server's agent environment, you get the error "command `puppet infrastructure recover_configuration` failed".

To fix this, install missing or incorrectly versioned gems on the primary server's agent environment. To find which gems are causing the error, check the backup logs for gem incompatibility issues with the error message. PE creates backup logs as a `report.txt` whenever you run a `puppet-backup` command.

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

The `puppet-backup restore` command fails with errors about a duplicate operator family

When restoring the `pe-rbac` database, the restore process exits with errors about a duplicate operator family, `citext_ops`.

To work around this issue:

1. Log into your PostgreSQL instance:

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

2. Run these commands:

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

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

Troubleshooting Windows

Troubleshoot Windows issues with failed installations and upgrades, and failed or incorrectly applied manifests. Use the error message reference to solve your issues. Enable debugging.

Installation fails

Check for these issues if Windows installation with Puppet fails.

The installation package isn't accessible

The source of an MSI or EXE package must be a file on either a local filesystem, a network mapped drive, or a UNC path.

RI-based installation sources aren't supported, but you can achieve a similar result by defining a file whose source is the primary server and then defining a package whose source is the local file.

Installation wasn't attempted with admin privileges

Puppet fails to install when trying to perform an unattended installation from the command line. A "norestart" message is returned, and installation logs indicate that installation is forbidden by system policy.

You must install as an administrator.

Upgrade fails

The Puppet MSI package overwrites existing entries in the `puppet.conf` file. If you upgrade or reinstall using a different primary server hostname, Puppet applies the new value in `$confdir\puppet.conf`.

When you upgrade Windows, you must use the same primary server hostname that you specified when you installed.

For information on what settings are preserved during an upgrade, see installing [Install Windows agents](#) on page 119.

Errors when applying a manifest or doing a Puppet agent run

Check for the following issues if manifests cannot be applied or are not applied correctly on Windows nodes.

Path or file separators are incorrect

For Windows, path separators must use a semi-colon (;), while file separators must use forward or backslashes as appropriate to the attribute.

In most resource attributes, the Puppet language accepts either forward or backslashes as the file separator. However, some attributes absolutely require forward slashes, and some attributes absolutely require backslashes.

When backslashes are double-quoted(""), they must be escaped. When single-quoted('), they can be escaped. For example, these are valid file resources:

```
file { 'c:\path\to\file.txt': }
file { 'c:\\path\\to\\file.txt': }
file { "c:\\path\\to\\file.txt": }
```

But this is an invalid path, because `\p`, `\t`, and `\f` are interpreted as escape sequences:

```
file { "c:\path\to\file.txt": }
```

For more information, see the [language reference about backslashes on Windows](#).

Cases are inconsistent

Several resources are case-insensitive on Windows, like files, users, groups. However, these resources can be case sensitive in Puppet.

When establishing dependencies among resources, make sure to specify the case consistently. Otherwise, Puppet can't resolve dependencies correctly. For example, applying this manifest fails, because Puppet doesn't recognize that ALEX and alex are the same user:

```
file { 'c:\foo\bar':
  ensure => directory,
  owner  => 'ALEX'
}
user { 'alex':
  ensure => present
}
...
err: /Stage[main]//File[c:\foo\bar]: Could not evaluate: Could not find user
ALEX
```

Shell built-ins are not executed

Puppet doesn't support a shell provider on Windows, so executing shell built-ins directly fails.

Wrap the built-in in `cmd.exe`:

```
exec { 'cmd.exe /c echo foo':
  path => 'c:\windows\system32;c:\windows'
}
```

Tip: In the 32-bit versions of Puppet, you might encounter file system redirection, where `system32` is switched to `sysWoW64` automatically.

PowerShell scripts are not executed

By default, PowerShell enforces a restricted execution policy which prevents the execution of scripts.

Specify the appropriate execution policy in the PowerShell command, for example:

```
exec { 'test':
  command => 'powershell.exe -executionpolicy remotesigned -file C:\test.ps1',
  path     => $::path
}
```

Or use the Puppet supported PowerShell.

Services are referenced with display names instead of short names

Windows services support a short name and a display name, but Puppet uses only short names.

1. Verify that your Puppet manifests use short names, for example `wuauserv`, not `Automatic Updates`.

Error messages

Use this reference to troubleshoot error messages when using Windows with Puppet.

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

This error occurs when you run the `puppet module` subcommand on newly provisioned Windows nodes. The Forge uses an SSL certificate signed by the GeoTrust Global CA certificate. Newly provisioned Windows nodes might not have that CA in their root CA store yet.

Download the "GeoTrust Global CA" certificate from GeoTrust's list of root certificates and manually install it by running `certutil -addstore Root GeoTrust_Global_CA.pem`.

- Service 'Puppet Agent' (`puppet`) failed to start. Verify that you have sufficient privileges to start system services.

This error occurs when installing Puppet on a UAC system from a non-elevated account. Although the installer displays the UAC prompt to install Puppet, it does not elevate privileges when trying to start the service.

Run from an elevated `cmd.exe` process when installing the MSI.

- Cannot run on Microsoft Windows without the `sys-admin`, `win32-process`, `win32-dir`, `win32-service` and `win32-taskscheduler` gems.

This error occurs if you attempt to run Windows without required gems.

Install specified gems: `gem install <GEM_NAME>`

- `"`
`err: /Stage[main]//Scheduled_task[task_system]: Could not evaluate: The operation completed successfully.`
`"`

This error occurs when using a the task scheduler gem earlier than version 0.2.1.

Update the task scheduling gem: `gem update win32-taskscheduler`

- `err: /Stage[main]//Exec[C:/tmp/foo.exe]/returns: change from notrun to 0 failed: CreateProcess() failed: Access is denied.`

This error occurs when requesting an executable from a remote primary server that cannot be executed.

Set the user/group executable bits appropriately on the primary server:

```
file { "C:/tmp/foo.exe":
  source => "puppet:///modules/foo/foo.exe",
}

exec { 'C:/tmp/foo.exe':
  logoutput => true
}
```

- `err: getaddrinfo: The storage control blocks were destroyed.`

This error occurs when the agent can't resolve a DNS name into an IP address or if the reverse DNS entry for the agent is wrong.

Verify that you can run `nslookup <dns>`. If this fails, there is a problem with the DNS settings on the Windows agent, for example, the primary dns suffix is not set. For more information, see [Microsoft's documentation on Naming Hosts and Domains](#).

- `err: Could not request certificate: The certificate retrieved from the master does not match the agent's private key.`

This error occurs if the agent's SSL directory is deleted after it retrieves a certificate from the primary server, or when running the agent in two different security contexts.

Elevate privileges using **Run as Administrator** when you select **Start Command Prompt with Puppet**.

- `err: Could not send report: SSL_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed. This is often because the time is out of sync on the server or client.`

This error occurs when Windows agents' time isn't synched. Windows agents that are part of an Active Directory domain automatically have their time synchronized with AD.

For agents that are not part of an AD domain, enable and add the Windows time service manually:

```
w32tm /register
net start w32time
w32tm /config /manualpeerlist:<ntpserver> /syncfromflags:manual /update
w32tm /resync
```

- `Error: Could not parse for environment production: Syntax error at '='; expected '}'`

This error occurs if `puppet apply -e` is used from the command line and the supplied command is surrounded with single quotes (`'`), which causes `cmd.exe` to interpret any `=>` in the command as a redirect.

Surround the command with double quotes (`"`) instead.

Logging and debugging

The Windows Event Log can be helpful when troubleshooting issues with Windows.

Enable Puppet to emit `--debug` and `--trace` messages to the Windows Event Log by stopping the Puppet service and restarting it:

```
c:\>sc stop puppet && sc start puppet --debug --trace
```

Note that this setting takes effect only until the next time the service is restarted, or until the system is rebooted.

Related information

[Logging for Puppet agent on Windows systems](#)

[Log files installed](#) on page 100

The software distributed with PE generates log files that you can collect for compliance or use for troubleshooting.