



Puppet Enterprise 2021.7.10

Contents

Welcome to Puppet Enterprise® 2021.7.10.....	7
PE software architecture.....	7
Component versions in recent PE releases.....	14
FIPS 140-2 enabled PE.....	17
Cloud PE.....	18
Getting support.....	21
Using the PE docs.....	28
Puppet platform documentation for PE.....	32
API index.....	33
Release notes.....	36
PE release notes.....	36
PE known issues.....	52
What's new since PE 2019.8.....	55
Getting started with Puppet Enterprise.....	65
Install PE.....	66
Install PE using installer tarball.....	66
Install PE using PIM.....	67
Log in to the PE console.....	72
Check the status of your primary server.....	72
Add nodes to the inventory.....	73
Add code and set up Code Manager.....	74
Manage Apache configuration on *nix targets.....	78
Install the apache module.....	79
Set up Apache node groups.....	80
Organize webserver configurations with roles and profiles.....	81
Manage IIS configuration on Windows targets.....	84
Install the iis module.....	85
Set up IIS node groups.....	86
Organize webserver configurations with roles and profiles.....	87
Next steps.....	91
Installing.....	91
Supported architectures.....	92
System requirements.....	96
Hardware requirements.....	96
Supported operating systems.....	98
Supported browsers.....	104
System configuration.....	104
What gets installed and where?.....	116
Installing PE.....	124
Install PE using the installer tarball.....	125
Install PE using PIM.....	140
Purchasing and installing a license key.....	144
Installing agents.....	145

Install agents with the install script.....	147
Install agents from the console.....	150
Install *nix agents.....	151
Install Windows agents.....	154
Install macOS agents.....	160
Install non-root agents.....	161
Managing certificate signing requests.....	164
Installing compilers.....	165
Installing client tools.....	172
Uninstalling.....	176
Upgrading.....	179
Upgrade paths.....	179
Upgrade cautions.....	180
Test modules before upgrade.....	182
Upgrading Puppet Enterprise.....	182
Upgrade PE using the installer tarball.....	183
Upgrade PE using PIM.....	191
Upgrading agents.....	196
Migrate PE.....	201
Configuring Puppet Enterprise.....	202
Tune infrastructure nodes.....	203
How to configure PE.....	211
Configure Puppet Server.....	215
Configure PuppetDB.....	220
Configure security settings.....	222
Configure proxies.....	228
Configure the console.....	230
Configure orchestration.....	235
Configure ulimit.....	240
Analytics data collection.....	241
Static catalogs.....	245
Configuring disaster recovery.....	248
Disaster recovery.....	248
Configure disaster recovery.....	258
Accessing the console.....	265
Reaching the console.....	265
Logging in.....	266
Managing access.....	268
User permissions and user roles.....	269
Creating and managing local users and user roles.....	278
LDAP authentication.....	281
Connecting LDAP external directory services to PE.....	281
Working with user groups from a LDAP external directory.....	287
SAML authentication.....	288
Connect a SAML identity provider to PE.....	289
Connect Microsoft ADFS to PE.....	295

Connect Okta to PE.....	300
Token-based authentication.....	303
RBAC API.....	310
Forming RBAC API requests.....	311
RBAC service errors.....	313
RBAC API v1.....	316
RBAC API v2.....	359
Activity service API.....	368
Forming activity service API requests.....	368
Event types reported by the activity service.....	370
Events endpoints.....	372
Monitoring and reporting.....	381
Monitoring infrastructure state.....	381
Viewing and managing packages.....	387
Value report.....	389
Infrastructure reports.....	393
Analyzing changes across Puppet runs.....	396
Puppet Enterprise metrics and status monitoring.....	399
View and manage Puppet Server metrics.....	401
Get started with Graphite.....	401
Available Graphite metrics.....	406
Metrics API.....	410
Metrics API v2.....	411
Metrics API v1.....	414
Status API.....	416
Status API authentication.....	418
Forming status API requests.....	418
Status API: services endpoint.....	419
Status API: services plaintext endpoint.....	424
Status API: metrics endpoint.....	425
Managing nodes.....	432
Adding and removing agent nodes.....	432
Adding and removing agentless nodes.....	433
How nodes are counted.....	437
Running Puppet on nodes.....	438
Grouping and classifying nodes.....	440
Making changes to node groups.....	449
Environment-based testing.....	451
Preconfigured node groups.....	455
Managing Windows nodes.....	459
Designing system configs (roles and profiles).....	486
The roles and profiles method.....	486
Roles and profiles example.....	490
Designing advanced profiles.....	493
Designing convenient roles.....	510
Node classifier API v1.....	513
Forming node classifier API requests.....	514
Groups endpoints.....	516
Classes endpoint.....	532
Classification endpoints.....	533
Commands endpoint.....	543
Environments endpoints.....	545

Nodes check-in history endpoint.....	548
Group children endpoint.....	551
Rules endpoint.....	555
Import hierarchy endpoint.....	555
Last class update endpoint.....	558
Update classes endpoint.....	558
Validation endpoint.....	559
Node classifier API errors.....	562
Node classifier API v2.....	563
Classification endpoints.....	563
Node inventory API v1.....	566
Forming node inventory API requests.....	566
Command endpoints.....	567
Query endpoints.....	571
Node inventory API errors.....	574
Managing patches.....	574
Configuring patch management.....	575
Patching nodes.....	581
Orchestrating Puppet runs, tasks, and plans.....	585
How Puppet orchestrator works.....	586
Setting up the orchestrator workflow.....	590
Configuring Puppet orchestrator.....	597
Run Puppet on demand.....	604
Run Puppet on demand from the console.....	604
Run Puppet on demand from the CLI.....	611
Tasks in PE.....	614
Installing tasks.....	615
Running tasks in PE.....	615
Writing tasks.....	626
Plans in PE.....	645
Plans in PE versus Bolt plans.....	646
Installing plans.....	648
Running plans in PE.....	649
Writing plans.....	652
Orchestrator API v1.....	678
Forming orchestrator API requests.....	678
Root endpoints.....	679
Command endpoints.....	681
Inventory endpoints.....	700
Jobs endpoints.....	703
Scheduled jobs endpoints.....	717
Plans endpoints.....	732
Plan jobs endpoints.....	736
Tasks endpoints.....	749
Usage endpoints.....	753
Scopes endpoints.....	755
Orchestrator API error responses.....	758
Migrating Bolt tasks and plans to PE.....	759
Managing and deploying Puppet code.....	762
Managing environments with a control repository.....	763

Managing environment content with a Puppetfile.....	767
Managing code with Code Manager.....	774
How Code Manager works.....	775
Set up Code Manager.....	778
Configure Code Manager.....	778
Configure Code Manager concurrency	783
Lockless code deploys.....	783
Customize Code Manager configuration in Hiera.....	785
Triggering Code Manager on the command line.....	795
Triggering Code Manager with a webhook.....	801
Triggering Code Manager with custom scripts.....	803
Troubleshooting Code Manager.....	804
Code Manager API.....	807
About file sync.....	817
Managing code with r10k.....	821
Set up r10k.....	822
Configure r10k.....	822
Customizing r10k configuration.....	823
Deploying environments with r10k.....	831
r10k command reference.....	834
SSL and certificates	836
Regenerate the console certificate.....	837
Regenerate the SAML certificate.....	838
Regenerate infrastructure certificates.....	838
Use an independent intermediate certificate authority.....	841
Use a custom SSL certificate for the console.....	843
Generate a custom Diffie-Hellman parameter file.....	845
Enable TLSv1.....	845
Maintenance	846
Back up and restore PE.....	846
Database maintenance.....	853
Rotating the inventory service secret key.....	854
Troubleshooting	855
Log locations.....	855
Troubleshooting installation.....	858
Troubleshooting disaster recovery.....	859
Troubleshooting puppet infrastructure run commands.....	859
Troubleshooting connections between components.....	860
Troubleshooting the databases.....	862
Troubleshooting cloud deployments.....	863
Authentication fails with SSH username or credentials.....	863
After 60 days, the puppetadmin user account stops working.....	864
Agent run fails for non-root users.....	864
Certificate-signing curl command has incorrect URL.....	864
Troubleshooting SAML connections.....	864
Troubleshooting backup and restore.....	865
Troubleshooting Windows.....	866

Welcome to Puppet Enterprise® 2021.7.10

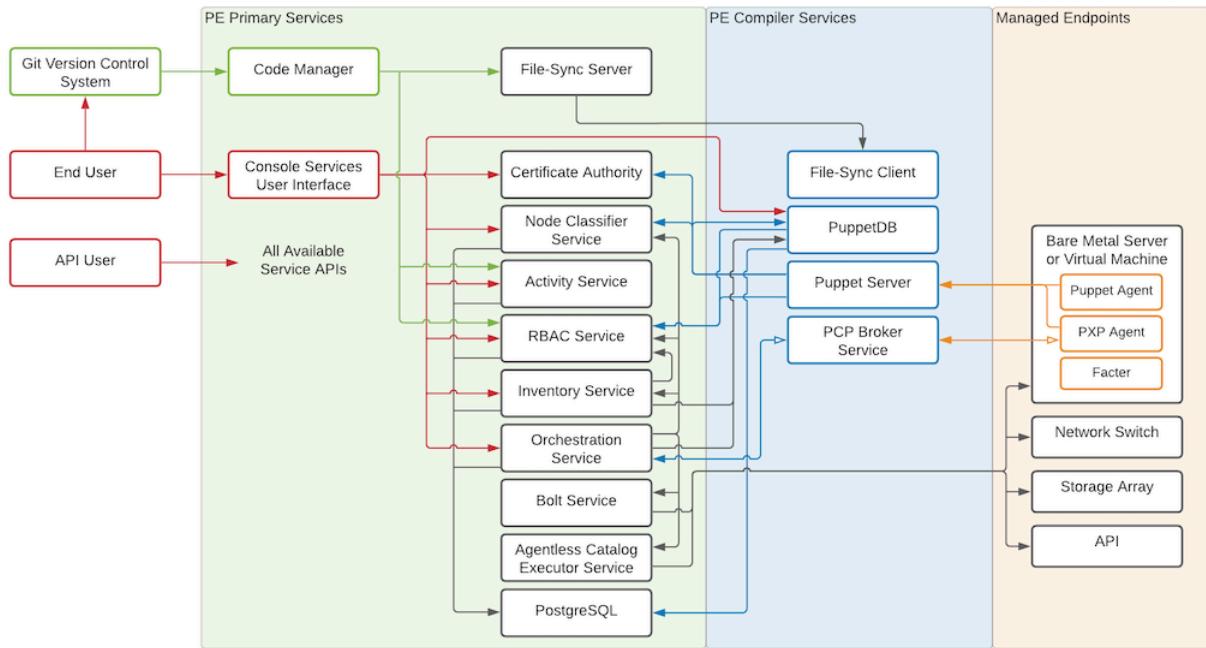
Puppet Enterprise (PE) helps you be productive, agile, and collaborative while managing your IT infrastructure. PE combines a model-driven approach with imperative task execution so you can effectively manage hybrid infrastructure across its entire lifecycle. PE provides the common language that all 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	Docs for related Puppet products
Release notes	Open source Puppet
Architecture overview	Continuous Delivery for Puppet Enterprise
System requirements	Puppet Comply
Getting started guide	Bolt
Install and configure PE	Puppet Development Kit
Install PE	Get help
Install agents	Support portal
Add agentless nodes	PE support lifecycle
Configure and tune PE	Archived PE docs
Manage your infrastructure	Share and contribute
Manage nodes	Puppet community
Run jobs, tasks, and plans	Puppet Forge
Deploy Puppet code	

PE software architecture

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

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



Related information

[Component versions in recent PE releases](#) on page 14

These tables show which components are in recent Puppet Enterprise (PE) long-term supported (LTS) releases and the prior LTS releases (2019.8.z). Component version tables for earlier releases are available in the [Documentation for other PE versions](#) on page 31. Component version tables for newer releases are available in the [latest documentation](#).

The primary server and compilers

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

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

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

Puppet Server

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

Catalog compiler

To configure a managed node, the agent uses a document called a catalog, which it downloads from the primary server or a compiler. The catalog describes the desired state for each resource on the node that you want to manage, and it can specify dependency information for resources that need to 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
- Accepts commands to sign or revoke certificates (optional)

The CA service uses CSPRNG-generated .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.

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

By default, the CA private key is located on the CA server at `cadir/ca_key.pem`. The default `cadir` is `/etc/puppetlabs/puppetserver/ca`. If you choose to use another directory, the key file must be stored in location readable by the `pe-puppet` user.

If you generate your own CA private key, the key must be RSA and the key file's PEM contents must begin with either BEGIN RSA PRIVATE KEY or BEGIN PRIVATE KEY. The entire CA chain must use the SHA-2 (or stronger) signing algorithm. Additionally, because the CA private key is one of the most critical files for security in your Puppet certificate infrastructure, the `pe-puppet` user must be the file owner and the permissions must be set to either mode: `0640` or `-rw-r-----`.

Related information

[Hardware requirements](#) on page 96

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 165

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 817

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

The Puppet agent

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

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

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

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

Puppet agent runs on *nix and Windows systems.

- [Puppet agent on *nix systems](#)
- [Puppet agent on Windows systems](#)

Facter

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

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

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

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

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

Console services

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

The console

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

The console can:

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

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

RBAC

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

For example:

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

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

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

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

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

Node classifier

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

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

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

You can also classify nodes using the NC API.

Related information

[Managing access](#) on page 268

Role-based access control (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 432

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 762

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

[Triggering Code Manager on the command line](#) on page 795

Use the `puppet-code` command to trigger Code Manager from the command line and deploy your environments.

[How Puppet orchestrator works](#) on page 586

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

Orchestration services

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

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

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

Agentless Catalog Executor (ACE) service

The ACE service enables you to run Puppet jobs, like tasks and plans, on nodes that don't have a Puppet agent installed on them. ACE service primarily runs through the orchestrator but it can be configured by itself. Go to [PE ACE server configuration](#) on page 603 to learn about configuring ACE.

Bolt vs ACE: Orchestrator uses both ACE and Bolt to run tasks and plans. While both can act on agentless targets, the primary difference is that Bolt server works with agentless nodes over WinRM or SSH, whereas ACE works with agentless devices, like network switches and firewalls, over other transports. Go to [PE Bolt server configuration](#) on page 602 to learn about how Bolt works in PE and configuring the Bolt server.

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 853

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

Security and communications

Puppet Enterprise (PE) services and components use a variety of communication and security protocols.

Service/Component	Communication Protocol	Authentication	Authorization
Puppet Server	HTTPS	SSL certificate verification with Puppet CA	trapperkeeper-auth
Certificate Authority	HTTPS	SSL certificate verification with Puppet CA	trapperkeeper-auth
Puppet agent	HTTPS	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 allow list
PostgreSQL	PostgreSQL TCP, SSL for PE	SSL certificate verification with Puppet CA	SSL certificate allow list
Activity service	HTTPS	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization
RBAC	HTTPS	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization

Service/Component	Communication Protocol	Authentication	Authorization
Classifier	HTTPS	SSL certificate verification with Puppet CA, token authentication	RBAC user-based authorization
Console Services UI	HTTPS	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 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

Compatible ciphers

Puppet Enterprise (PE) is compatible with a variety of ciphers for different services.

Most TLSv1.2 ciphers are available in IANA or OpenSSL format, depending on the service it's used for. You can use the same TLSv1.3 ciphers interchangeably for OpenSSL and IANA formats.

Use IANA format for these services on TLSv1.2:

Puppet Server

PuppetDB

Console services

Orchestrator

Use OpenSSL format for these services on TLSv1.2:

Bolt Server

ACE server

PostgreSQL

NGINX

Restriction:

To use TLSv1.3, you must enable both TLSv1.2 and TLSv1.3.

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

The following table describes the default TLSv1.2 and TLSv1.3 ciphers PE accepts for FIPS and non-FIPS installations. If you use an unsupported cipher, it is rejected when the service tries to establish a connection.

Ciphers in IANA format	Ciphers in OpenSSL format	TLS protocol	FIPS support
TLS_AES_256_GCM_SHA384	AES_256_GCM_SHA384	TLSv1.3	Yes

Ciphers in IANA format	Ciphers in OpenSSL format	TLS protocol	FIPS support
TLS_AES_128_GCM_SHA256	AES_GCM_SHA256	TLSv1.3	Yes
TLS_CHACHA20_POLY1305	CHACHA20_POLY1305_SHA356	TLSv1.3	No
Restriction: Only for Bolt server, ACE server, and NGINX	Restriction: Only for Bolt server, ACE server, and NGINX		
N/A	ECDHE-ECDSA-CHACHA20-POLY1305 Restriction: Only for NGINX	TLSv1.2	No
N/A	ECDHE-RSA-CHACHA20-POLY1305 Restriction: Only for NGINX	TLSv1.2	No
TLS_ECDHE_ECDSA_WITH_AES128_GCM_SHA256	AES128-GCM-SHA256	TLSv1.2	Yes
TLS_ECDHE_ECDSA_WITH_AES256_GCM_SHA384	AES256-GCM-SHA384	TLSv1.2	Yes
TLS_ECDHE_RSA_WITH_AES128_GCM_SHA256	AES128-GCM-SHA256	TLSv1.2	Yes
TLS_ECDHE_RSA_WITH_AES256_GCM_SHA384	AES256-GCM-SHA384	TLSv1.2	Yes
TLS_DHE_RSA_WITH_AECDH256_GCM_SHA384	DHE256_GCM_SHA384	TLSv1.2	No
TLS_DHE_RSA_WITH_AECDH256_GCM_SHA256	DHE256_GCM_SHA256	TLSv1.2	No

Related information

[FIPS 140-2 enabled PE](#) on page 17

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.

Component versions in recent PE releases

These tables show which components are in recent Puppet Enterprise (PE) long-term supported (LTS) releases and the prior LTS releases (2019.8.z). Component version tables for earlier releases are available in the [Documentation for other PE versions](#) on page 31. Component version tables for newer releases are available in the [latest documentation](#).

Puppet Enterprise agent and server components

The following table shows the components that are installed on all agent nodes.

Tip: 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	Factor	Ruby	OpenSSL
2021.7.10	7.35.0	4.8.0	<ul style="list-style-type: none"> • MRI Ruby: 2.7.8 (Puppet agent) • JRuby: 9.3.14.0 (Puppet server) 	1.1.1v
2021.7.9	7.32.1	4.8.0	<ul style="list-style-type: none"> • MRI Ruby: 2.7.8 (Puppet agent) • JRuby: 9.3.14.0 (Puppet server) 	1.1.1v
2021.7.8	7.30.0	4.7.0	2.7.8	1.1.1v
2021.7.7	7.28.0	4.5.2	2.7.8	1.1.1v
2021.7.6	7.27.0	4.5.1	2.7.8	1.1.1v
2021.7.5	7.26.0	4.4.3	2.7.8	1.1.1v
2021.7.4	7.24.0	4.3.1	2.7.7	1.1.1t
2021.7.3	7.24.0	4.3.1	2.7.7	1.1.1t
2021.7.2	7.21.0	4.2.14	2.7.7	1.1.1q
2021.7.1	7.20.0	4.2.13	2.7.6	1.1.1q
2021.7.0	7.18.0	4.2.11	2.7.6	1.1.1q
2021.6	7.16.0	4.2.8	2.7.6	1.1.1n
2021.5	7.14.0	4.2.7	2.7.5	1.1.1l
2021.4	7.12.1	4.2.5	2.7.3	1.1.1l
2021.3	7.9.0	4.2.2	2.7.3	1.1.1k
2021.2	7.8.0	4.2.1	2.7.3	1.1.1k
2021.1	7.6.1	4.1.1	2.7.3	1.1.1i
2021.0	7.4.1	4.0.51	2.7.2	1.1.1i
2019.8.12	6.28.0	3.14.24	2.5.9	1.1.1q
2019.8.11	6.27.0	3.14.23	2.5.9	1.1.1n
2019.8.10	6.26.0	3.14.22	2.5.9	1.1.1l
2019.8.9	6.25.1	3.14.21	2.5.9	1.1.1l
2019.8.8	6.24.0	3.14.19	2.5.9	1.1.1k
2019.8.7	6.23.0	3.14.18	2.5.9	1.1.1k
2019.8.6	6.22.1	3.14.17	2.5.9	1.1.1g
2019.8.5	6.21.1	3.14.16	2.5.8	1.1.1i
2019.8.4	6.19.1	3.14.14	2.5.8	1.1.1g
2019.8.3	6.19.1	3.14.14	2.5.8	1.1.1g
2019.8.1	6.17.0	3.14.12	2.5.8	1.1.1g
2019.8	6.16.0	3.14.11	2.5.8	1.1.1g

The following table shows components that are installed on server nodes.

PE Version	Puppet Server	PuppetDB	r10k	Bolt Services	Agentless Catalog Executor (ACE) Services	PostgreSQL	Java	Nginx
2021.7.10	7.17.4	7.21.0	3.16.2	3.30.0	1.2.4	14.13	11.0.26.4.26.2	
2021.7.9	7.17.2	7.19.1	3.16.2	3.30.0	1.2.4	14.13	11.0.24.8.26.2	
2021.7.8	7.17.1	7.18.0	3.16.1	3.29.0	1.2.4	14.11	11.0.23.9.25.1	
2021.7.7	7.15.0	7.16.0	3.16.0	3.27.4	1.2.4	14.10	11.0.22.7.25.1	
2021.7.6	7.14.0	7.15.0	3.16.0	3.27.4	1.2.4	14.8	11.0.21.9.25.1	
2021.7.5	7.13.1	7.14.0	3.16.0	3.27.2	1.2.4	14.8	11.0.20.8.25.1	
2021.7.4	7.11.0	7.13.0	3.15.4	3.27.1	1.2.4	14.5	11.0.19.6.22.0	
2021.7.3	7.11.0	7.13.0	3.15.4	3.27.1	1.2.4	14.5	11.0.19.6.22.0	
2021.7.2	7.9.4	7.12.1	3.15.4	3.26.2	1.2.4	14.5	11.0.17.8.22.0	
2021.7.1	7.9.2	7.11.2	3.15.2	3.26.1	1.2.4	14.5	11.0.6 1.22.0	
2021.7.0	7.9.0	7.11.1	3.15.1	3.26.1	1.2.4	14.5	11.0 1.22.0	
2021.6	7.7.0	7.10.1	3.14.0	3.22.1	1.2.4	14.1	11.0 1.21.0	
2021.5	7.6.0	7.9.2	3.14.0	3.21.0	1.2.4	11.13	11.0 1.21.0	
2021.4	7.4.2	7.7.1	3.13.0	3.20.0	1.2.4	11.13	11.0 1.21.0	
2021.3	7.2.1	7.5.2	3.10.0	3.13.0	1.2.4	11.13	11.0 1.21.0	
2021.2	7.2.0	7.4.1	3.9.2	3.10.0	1.2.4	11.11	11.0 1.21.0	
2021.1	7.1.2	7.3.1	3.9.0	3.7.1	1.2.4	11.11	11.0 1.19.6	
2021.0	7.0.3	7.1.0	3.8.0	3.0.0	1.2.2	11.10	11.0 1.19.6	
2019.8.12	6.20.0	6.22.1	3.15.1	3.25.0	1.2.4	11.17	11.0 1.22.0	
2019.8.11	6.19.0	6.21.0	3.14.0	3.22.1	1.2.4	11.15	11.0 1.21.0	
2019.8.10	6.18.0	6.20.2	3.14.0	3.21.0	1.2.4	11.13	11.0 1.21.0	
2019.8.9	6.17.1	6.19.1	3.13.0	3.20.0	1.2.4	11.13	11.0 1.21.0	
2019.8.8	6.16.1	6.18.2	3.10.0	3.13.0	1.2.4	11.13	11.0 1.17.0	
2019.8.7	6.16.0	6.17.0	3.9.2	3.10.0	1.2.4	11.11	11.0 1.21.0	
2019.8.6	6.15.3	6.16.1	3.9.0	3.7.1	1.2.4	11.11	11.0 1.19.6	
2019.8.5	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	

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.

Restriction: Puppet Server 6.x is no longer developed or tested.

Agent	Server		
	Puppet 6.x PE 2019.1 through 2019.8	Puppet 7.x PE 2021.0 through 2023.2	Puppet 8.x PE 2023.4 and later
6.x	#	#	#
7.x		#	#
8.x			#

Task compatibility

Information is provided about the Puppet task specification that is compatible with Puppet Enterprise (PE).

PE 2021.7.10 supports version 1, revision 4 of the Puppet task specification.

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 operating system](#) 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 an agent's vendored 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.

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

Related information

[Supported operating systems and devices](#) on page 98

You can install PE and the agent on these supported platforms.

[Installing PE](#) on page 124

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

Cloud PE

Puppet Enterprise (PE) 2021.7 cloud images are available from Amazon Web Services (AWS) and Microsoft Azure.

Cloud images contain a *standard installation* with PE services running on a primary server installed in your chosen cloud environment. You can use cloud images to manage deployments of up to 2,500 nodes.

Restriction: Avoid cloud deployments with more than 2,500 nodes because cloud PE does not support compilers.

After launching a cloud image, you can use PE as you would any on-premises standard installation.

To get started with cloud PE, review the following information.

Cloud providers

PE cloud images are available from the following providers:

- [AWS Marketplace](#)
- [Microsoft Azure Marketplace](#)

Licensing

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

System requirements

For system requirements, see [Hardware requirements for cloud deployments](#) on page 97.

Security groups

For security information, see [Firewall configuration for standard installations](#) on page 106. For a sample security group policy in JavaScript Object Notation (JSON) format for Amazon Elastic Compute Cloud (EC2), see [Example EC2 security group policy](#).

Identity and access management

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

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

Installing

For installation instructions, see [Installing in cloud environments](#).

Connecting agents

Cloud images contain agent packages for all [supported operating systems](#). You can install agents by using any supported method. For instructions, see [Installing agents](#) on page 145.

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

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

Configuring DNS

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

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

Tuning

PE cloud images are tuned by using default settings for standard installations. You can tune your primary server in cloud deployments as you would any on-premises installation. For instructions, see [Tune infrastructure nodes](#) on page 203.

Running commands on nodes

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

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

Scaling

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

- AWS – [Change the instance type](#)
- Azure – [Use the portal to attach a data disk to a Linux VM](#)

Upgrading

To upgrade a cloud deployment, follow the instructions in [Upgrade a standard installation](#) on page 183.

Troubleshooting

In case of issues, see [Troubleshooting cloud deployments](#) on page 863.

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

Example EC2 security group policy

The following JavaScript Object Notation (JSON) structure provides an example of security group policy for Amazon Elastic Compute Cloud (EC2). The policy accommodates inbound network ports required by Puppet.

```
{
  "IpPermissions": [
    {
      "PrefixListIds": [],
      "FromPort": 22,
      "IpRanges": [{"CidrIp": "0.0.0.0/0"}],
      "ToPort": 22,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 443,
      "IpRanges": [{"CidrIp": "0.0.0.0/0"}],
      "ToPort": 443,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 8140,
      "IpRanges": [{"CidrIp": "<SUBNET-CIDR>"}],
      "ToPort": 8140,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 8142,
      "IpRanges": [{"CidrIp": "<SUBNET-CIDR>"}],
      "ToPort": 8142,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 8143,
      "IpRanges": [{"CidrIp": "<SUBNET-CIDR>"}],
      "ToPort": 8143,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    },
    {
      "PrefixListIds": [],
      "FromPort": 61613,
      "IpRanges": [{"CidrIp": "<SUBNET-CIDR>"}],
      "ToPort": 61613,
      "IpProtocol": "tcp",
      "UserIdGroupPairs": []
    }
  ],
  "IpPermissionsEgress": [
    {
      "IpProtocol": "-1",
      "IpRanges": [{"CidrIp": "0.0.0.0/0"}],
      "UserIdGroupPairs": [],
      "PrefixListIds": []
    }
  ]
}
```

```

        }
    ]
}
```

Getting support

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

Puppet Enterprise support life cycle

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

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

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

If the latest release with the most up-to-date features is right for you, [download or try](#) the latest PE release, or download an older supported release from the [Previous Releases](#) page. We recommend following our [Installing](#) on page 91 guide and understanding the [System requirements](#) on page 96 before downloading the installation package.

Open source tools and libraries

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

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

Support portal

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

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

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

Puppet metrics collector

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

The Puppet metrics collector is packaged in a module that is installed with PE. By default, the module collects Puppet services metrics and does not collect system metrics. You can enable and disable metrics collection by setting Boolean values for these parameters.

- `puppet_enterprise::enable_metrics_collection`
- `puppet_enterprise::enable_system_metrics_collection`

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

Related information

[How to configure PE](#) on page 211

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. For example, you might want to add your certificate to the allowlist, increase the max-threads setting for `http` and `https` requests, or configure the number of JRuby instances.

[Puppet Enterprise metrics and status monitoring](#) on page 399

You can use Puppet Enterprise (PE) metrics and status monitoring for your own performance tuning or provide the information to Support for troubleshooting.

PE support script

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

The `pe_support_script` module, bundled with the installer, provides the script.

Running the support script

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

PE version 2021.4 includes version 3 of the support script. Version 3 has more options that can be used to modify the support script behavior. As such, some options in version 3 are not available in combination with the `--v1` option. This is because the `--v1` option activates the legacy (version 1) support script. Options not compatible with version 1 are designated in the table below as *Not compatible with the --v1 parameter*.

These options can modify the support script output:

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 PATH 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. Default is 7 (seven days).

Option	Description
--v1	Activate version 1 of the support script. This option is not compatible with options designated as <i>Not compatible with the --v1 parameter</i> .
--v3	Activate version 3 of the support script. This is a default but can be overridden with --v1.
--list	List diagnostics that can be enabled or disabled. Diagnostics labeled "opt-in" must be explicitly enabled. All others are enabled by default. <i>Not compatible with the --v1 parameter</i> .
--enable <LIST>	A comma-separated list of diagnostic names to enable. Use the --list option to print available names. The --enable option must be used to activate diagnostics marked as "opt-in." <i>Not compatible with the --v1 parameter</i> .
--disable <LIST>	A comma-separated list of diagnostic names to disable. Use the --list option to print available names. <i>Not compatible with the --v1 parameter</i> .
--only <LIST>	A comma-separated list of diagnostic names to enable. All other diagnostics are disabled. Use the --list option to print available names. <i>Not compatible with the --v1 parameter</i> .
--upload	Upload the output tarball to Puppet Support via SFTP. Requires the --ticket <NUMBER> option to be used. <i>Not compatible with the --v1 parameter</i> .
--upload_disable_host_key_check	Disable SFTP host key checking. Go to Use SFTP to upload files to Puppet Support for a list of current host key values. <i>Not compatible with the --v1 parameter</i> .
--upload_user <USER>	Specify a SFTP user to use when uploading. If not specified, a shared write-only account is used. <i>Not compatible with the --v1 parameter</i> .
--upload_key <FILE>	Specify a SFTP key to use with --upload_user. <i>Not compatible with the --v1 parameter</i> .

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

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

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

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

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

Information collected by the support script

This information is collected by the support script.

base-status

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

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

system

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

The `system.config` check collects:

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

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

The `system.status` check collects:

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

puppet-agent

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

The `puppet.config` check collects:

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

The `puppet-agent.logs` check collects:

- Puppet log files from `/var/log/puppetlabs/puppet`
- JournalD logs for the puppet service and ppxp-agent service
- PXP agent log files from `/var/log/puppetlabs/puppet`

The `puppet-agent.status` check collects:

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

puppetserver

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

The `puppetserver.config` check collects these Puppet Server configuration files:

- `/etc/puppetlabs/code/hiera.yaml`
- `/etc/puppetlabs/puppet/auth.conf`
- `/etc/puppetlabs/puppet/autosign.conf`
- `/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`

The `puppetserver.logs` check collects:

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

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

The `puppetserver.status` check collects:

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

puppetdb

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

The `puppetdb.config` check collects these configuration files:

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

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

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

The `puppetdb.status` check collects:

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

pe

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

The `pe.config` check collects:

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

The `pe.logs` check collects:

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

The `pe.status` check collects output from `puppet infra status`, current tuning settings from `puppet infra tune`, and recommended tuning settings from `puppet infra tune`.

The `pe.file-sync` check is disabled by default. When activated by the `--enable` option, this check collects:

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

pe.console

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

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

- `/etc/puppetlabs/console-services/bootstrap.cfg`
- `/etc/puppetlabs/console-services/conf.d/`

- /etc/puppetlabs/console-services/logback.xml
- /etc/puppetlabs/console-services/rbac-certificate-whitelist
- /etc/puppetlabs/console-services/request-logging.xml
- /etc/puppetlabs/nginx/conf.d/
- /etc/puppetlabs/nginx/nginx.conf

The `pe.console.logs` check collects :

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

The `pe.console.status` check collects:

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

The `pe.console.classifier-groups` check is disabled by default. When activated by the `--enable` option, the `pe.console.classifier-groups` check collects all classification data provided by the `/v1/groups` API endpoint.

pe.orchestration

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

The `pe.orchestration.config` check collects:

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

The `pe.orchestration.logs` check collects:

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

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

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

pe.postgres

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

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

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

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

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

The `pe.postgres.status` check collects:

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

Community support

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

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

Using the PE docs

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

Using example commands

These guidelines can help you understand and customize the example commands you'll find in the Puppet Enterprise (PE) docs.

Ports, paths, and other input

Some examples in the PE docs use `puppet` commands to populate variables and `curl` arguments. This can take the guesswork out of providing those values. In the following example, the `puppet config print server` command supplies the DNS name for a `curl` command:

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

```
curl "$url"
```

In these examples, puppet commands generate cert paths:

```
--cert $(puppet config print --section main hostcert) \
--key $(puppet config print --section main hostprivkey) \
--cacert $(puppet config print --section main localcacert) \
```

puppet commands can return different values depending on various conditions. Make sure you run the entire example (including commands setting environment variables and the curl command) as the root, administrator, or with equivalent [elevated privileges](#).

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

Tip: If an example command uses a service's default port, and you changed the service's port, you must change the port number accordingly in the command.

Authentication tokens in curl commands

If a curl command requires token-based authentication, the example might contain this line:

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

If you have an actual authentication token available, you can use that in the command instead, such as:

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

For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

Modifications for Windows

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

Tip: With the exception of Windows-specific commands, code samples use backslashes (\) as line-continuation characters. In Windows, the equivalent characters are carets (^) and, for PowerShell specifically, backticks (`).

Additionally, *nix commands use forward slashes (/) as directory separator characters. You might use either backslashes or forward slashes as directory separator characters in your Windows commands; however some modules and commands require you to use one or the other. For modules, check the module's [Forge](#) page for information about Windows modifications or requirements.

Furthermore, Windows commands might require wrapping strings or arguments in double quotes rather than single quotes.

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

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

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

Native curl	PowerShell equivalent
curl	Invoke-WebRequest

Native curl	PowerShell equivalent
-k or --insecure	[System.Net.ServicePointManager]::ServerCertificateValidationCallback = \$true
-H	-Headers
-X	-Method
-d	-Body
\ (as a line-continuation character)	`

You can learn more about `Invoke-WebRequest` and the arguments it accepts in the [Microsoft PowerShell documentation](#). You can also learn about [running Puppet language commands on Windows](#) in the Puppet documentation.

Related information

[Executing PowerShell code](#) on page 468

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.

[Managing Windows nodes](#) on page 459

You can use Puppet Enterprise (PE) to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks with modules from the Forge.

[Configuring patch management](#) on page 575

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.

[Troubleshooting Windows](#) on page 866

Troubleshoot issues in Windows PE installations, such as failed installations, failed upgrades, problems applying manifests, and other issues.

Commands with elevated privileges

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

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

root (or administrator)

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

sudo

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

Run as administrator or runas

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

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

Restriction: You must log in as the root user (or administrator) to run `puppet infrastructure` commands.

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

Documentation for other PE versions

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

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

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

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

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

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

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

PE Version	URL
2023.8.2 (LTS)	https://www.puppet.com/docs/pe/2023.8/pe_user_guide.html
2021.7.z	https://www.puppet.com/docs/pe/2021.7/pe_user_guide.html
2021.6	https://puppet.com/docs/pe/2021.6/pe_user_guide.html
2021.5	https://puppet.com/docs/pe/2021.5/pe_user_guide.html
2021.4	https://puppet.com/docs/pe/2021.4/pe_user_guide.html
2021.3	https://puppet.com/docs/pe/2021.3/pe_user_guide.html
2021.2	https://puppet.com/docs/pe/2021.2/pe_user_guide.html
2021.1	https://puppet.com/docs/pe/2021.1/pe_user_guide.html
2021.0	PE docs archive on GitHub
2019.8.z (EOL)	Documentation for the most-recent incremental release is available at: https://puppet.com/docs/pe/2019.8/pe_user_guide.html For earlier incremental releases, go to the PE docs archive on GitHub .
2019.7	https://puppet.com/docs/pe/2019.7/pe_user_guide.html
Earlier versions	PE docs archive on GitHub

Puppet platform documentation for PE

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

Puppet

Puppet 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 documentation, which is separate from the [Puppet documentation](#). Use the navigation sidebar to [navigate the documentation](#).

Important: The Puppet documentation has information about installing the open source release of Puppet. PE users can ignore those pages.

These are good starting points for getting familiar with Puppet:

Language

- Fundamental pieces of [the Puppet language](#) are [resources](#), [variables](#), [conditional statements and expressions](#), and [relationships and ordering](#).
- You use [Classes](#) and [defined resource types](#) to organize Puppet code into useful chunks. Classes are the main unit of Puppet code you'll interact with on a daily basis. You can assign classes to nodes in the PE console.
- You can use [facts and built-in variables](#) as special variables in your Puppet manifests.

Modules

- Most Puppet code goes in modules. [Modules overview](#) explains how modules work.
- Learn how to [install modules](#) from the Forge and [publish modules](#) on the Forge.
- Use PE's code management features to control your modules instead of installing manually, as explained in [Managing and deploying Puppet code](#) on page 762.

Services and commands

- Learn about [Puppet commands](#), including an overview of Puppet's architecture and a list of the main Puppet commands and services you'll interact with.
- You can also learn about [running Puppet commands on Windows](#).

Built-in resource types and functions

- The [resource type reference](#) describes built-in Puppet resource types.
- The [built-in function reference](#) describes built-in Puppet functions.

Important directories and files

- Most Puppet content goes in [environments](#).
- The [codedir](#) is the main directory for Puppet code and data, and the [confdir](#) contains Puppet config files.
- The [modulepath](#) and the [main manifest](#) depend on the current environment.

Configuration

- The main config file for Puppet is `/etc/puppetlabs/puppet/puppet.conf`. You can learn about [Puppet settings](#) and [puppet.conf](#).
- Other [Puppet config files](#) are used for special purposes.

Puppet Server

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.

Generally, PE users don't need to directly manage Puppet Server, and the [Puppet documentation](#) describes how Puppet Server evaluates the Puppet language and loads environments and modules. For users who need to access the [environment cache](#) and [JRuby pool](#) administrative APIs, you can find background information in the rest of the Puppet Server docs.

Note: The Puppet Server documentation has information about installing the open source release of Puppet Server. PE users can ignore those pages. Additionally, a built-in Puppet module manages the Puppet Server config files; to change most settings, you can set the appropriate class parameters in the console.

Facter

[Facter](#) is a system profiling tool that Puppet agent uses it to send important system information to Puppet Server. Puppet Server can access that information when compiling that node's catalog.

- For a list of variables you can use in your code, check the [Facter: Core facts reference](#).
- You can write your own custom facts, as explained in [Writing custom facts](#) and the [Custom facts overview](#).

Hiera

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

Start with [About Hiera](#), and then use the navigation sidebar to explore the Hiera docs.

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](#) 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 use the query API. Here are some resources to get you started:

- The [API query tutorial](#) walks you through 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.

Important: The [PuppetDB documentation](#) has information about installing the open source release of PuppetDB. PE user can ignore those pages.

Related information

[Managing and deploying Puppet code](#) on page 762

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

API index

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

Puppet Enterprise APIs

For information on port requirements, see [System configuration](#) on page 104.

API	Useful for
Node inventory API v1 on page 566	<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.
RBAC service API v1	<ul style="list-style-type: none"> • Managing access to PE. • 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"> • Fetch users (with filters). • Revoking authentication tokens. • Validating new user groups against LDAP. • Viewing information about an LDAP connection.
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.

API	Useful for
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
Server-specific Puppet API	<ul style="list-style-type: none"> Environment classes Environment modules Static file content
Puppet Server status API	<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. Checking the state, memory usage, and uptime of the services running on Puppet Server.
Puppet Server metrics API	<ul style="list-style-type: none"> Querying Puppet Server performance and usage metrics. The <code>/metrics/v1/mbeans</code> endpoint is deprecated.
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.

API	Useful for
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® 2021.7.

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.

Tip: This version of documentation represents the most recent update in this release stream. Features and functionality might differ from previous releases in this stream.

Security and vulnerability announcements are posted at [Security: Puppet's Vulnerability Submission Process](#).

- [PE release notes](#) on page 36

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

- [PE known issues](#) on page 52

These are the known issues in PE 2021.7.

- [What's new since PE 2019.8](#) on page 55

This page describes the new features, enhancements, deprecations, and other notable changes since the previous LTS release (2019.8), specifically PE versions 2021.0 through 2021.7.0. The previous LTS release stream comprised PE versions 2019.8.0 through 2019.8.12.

PE release notes

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

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

PE 2021.7.10

Released February 2025

Important: PE 2021.7 is our previous long-term supported release; it reached End-of-Life (EOL) on February 28, 2025. PE 2023 is our PE LTS series. PE 2025 is the leading-edge PE series.

- To access End-of-Life (EOL) dates and maintenance information, see [PE End-of-Life \(EOL\)](#).
- For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023 documentation](#).
- Customers on 2021.7.z are encouraged to [upgrade to 2023.8.z](#).

Note: To access the release notes for the Puppet® platform, including Puppet agent, Puppet Server, Facter, and PuppetDB, see [Platform release notes](#).

Platform support

In PE 2021.7.10, support is added for the following operating system platforms:

Agent platforms added

This release adds support for the Puppet agent on the following operating system platforms:

- Microsoft Windows Server 2016 FIPS

Security fixes

Addressed the following CVEs:

- CVE-2024-49761
- CVE-2024-27281

Resolved issues

Puppet code status command no longer fails to run

In PE 2021.7.8-2021.7.9, PE 2023.7.0-2023.8.1, and PE 2025.0.0, Puppet code status command failed to run.

This issue is fixed in PE 2021.7.10, 2023.8.2 and 2025.1.0.

PE 2021.7.9

Released August 2024

Important: PE 2021.7 is in overlap support until 28th February, 2025 and PE 2023 is our PE LTS series. PE 2025.0 is the leading-edge PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023 documentation](#).

Note: To access the release notes for the Puppet® platform, including Puppet agent, Puppet Server, Facter, and PuppetDB, see [Platform release notes](#).

Enhancements

Default to find reports generated within the last 30 minutes on the Events screen in the PE console

In order to make the page load faster and be more efficient, the Events screen in the PE console has changed the default period from *Events from the last run* to *Events in the last 30 minutes*.

Usage of find and chown in lockless Puppet code improved

A slow and I/O intensive operation in compiler catalogs (codedirs chown) is now optional and may be disabled with the `puppet_enterprise::master::file_sync::chown_code_to_pe_puppet` parameter.

Legacy facts disabled while running 2021.7.x (maintenance) to help improve user's upgrade testing

While there are other methods available to test if your code base is Puppet 8 compatible, disabling legacy facts in Puppet Enterprise ensures there's no issues/blockers. This process is not possible in earlier versions of PE 2021.7.x because legacy facts is still used. Legacy facts has been removed in version PE 2021.7.9.

Platform support

In PE 2021.7.9, support is added for the following operating system platforms:

Agent platforms added

This release adds support for the Puppet agent on the following operating system platforms:

- RedHat Enterprise Linux 9 ppc64le

- Fedora 40 x86_64
- Ubuntu 24.04 amd64
- Ubuntu 24.04 aarch64
- Amazon Linux 2 aarch64
- Rocky 9 x86_64
- Rocky 9 aarch64
- Alma Linux 9 x86_64
- Alma Linux 9 aarch64

Resolved issues

Tasks containing a description without any parameters fixed

In PE 2023.7 and PE 2021.7.8, if the task metadata on the **Run a task** screen in the PE console, contained a description without any parameters, the console did not display the description. This issue has been resolved in PE 2023.8.0 and PE 2021.7.9.

SAML login no longer fails when changing the `rbac_token_maximum_lifetime` class

When modifying the `rbac_token_maximum_lifetime` parameter in **Node groups > PE Infrastructure** in the PE console to anything other than the default of 10y, the user received the following error when trying to use SAML login:

```
{
  "kind": "puppetlabs.rbac/saml-response-processing-error",
  "msg": "There was an error processing the SAML response: \"No
implementation of method: :to-date-time of protocol: #'clj-time.coerce/
ICoerce found for class: clojure.lang.Keyword\""
}
```

This issue is fixed in PE 2023.8.0 and PE 2021.7.9.

Exec resources failure while using lockless code deploy and applying a compiler's catalog simultaneously fixed

A race condition that could cause one or more executive resources to fail if a code deploy occurred at the same time as a compiler's catalog was applied has been fixed.

Reliability of the `toggle_lockless_deploys` plan fixed

In versions PE 2023.7 and PE 2021.7.8, the `toggle_lockless_deploys` plan could encounter a race condition when running causing spurious failures. It also would not update Hiera data in the way needed for the lockless deploys setting to be honored on the replica in DR/HA setups. The plan is now more robust and works with DR/HA.

Unable to view a node's Groups tab in the PE console if view permission is not enabled for any single group the node is in fixed

In versions PE 2023.7 and PE 2021.7.3 - 2021.7.8, if a user did not have permission to view some of the groups their node were in, they could not view their node in any of their node's groups to which they have rights and received an error message stating that they did not have permission to view the group. This issue has been resolved in PE 2023.8 and PE 2021.7.9.

Occasional failure due to a race condition while provisioning a replica fixed

During provisioning of a replica, with either the `puppet infra provision replica` or `puppet infra run enable_ha_failover` commands, when the subscription on the replica was established, the Puppet agent did not wait for the subscription initialization to complete and let it run in the background. This resulted in a race condition in which pglogical performed a `pg_restore` on the database structure while the Puppet agent simultaneously made other database changes. This caused a variety of error signatures, but typically displayed as `ERROR: tuple concurrently updated` in the PostgreSQL log. Now, the provisioning process waits for the database structure and data to complete its initial sync before proceeding. If you have a large pe-activity database, this may cause provisioning to take a bit longer than usual, up to 10 extra minutes.

PE 2021.7.8

Released May 2024

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the leading-edge PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.y documentation](#).

New features

Experience the full value of Puppet Enterprise

If you have installed Puppet Enterprise, you can separately install and use Security Compliance Management (formerly Puppet Comply®) and Continuous Delivery, which are both now covered by your Puppet Enterprise license. You can also [contact our sales team](#) to enable the following additional premium features:

- Security Compliance Enforcement (formerly CEM)
- Advanced Impact Analysis capabilities within Continuous Delivery

Enhancements

Feature toggle for lockless code deploys

If you have enabled Code Manager, you can now turn the lockless code deploys feature on or off by running a `puppet infra` plan on your primary server. See [Toggle lockless code deploys on or off](#) on page 785.

Disaster recovery workflows improved

This release includes improvements to disaster recovery workflows for standard and large installations. The enhancements help to ensure smooth failover to your primary server replica, and minimize potential for disruption in cases where replica promotion is required. See [Configuring disaster recovery](#) on page 248.

Correct CA directory automatically set up during upgrade

Starting in 2021.7.8 and 2023.7, when you upgrade PE, the installer checks that your certificate authority (CA) directory is set up at `/etc/puppetlabs/puppetserver/ca` and if necessary, the installer automatically migrates the CA to this directory. This enhancement mitigates the risk of certificate collisions during disaster recovery procedures.

Enhanced logging of schema validation

In the Puppet Server version bundled with PE 2021.7.8, validation messages in the logs have been improved to provide more context about failed schemas.

Platform support

In PE 2021.7.8, support is added for the following operating system platforms:

Agent platforms added

This release adds support for the Puppet agent on the following operating system platforms:

- Amazon Linux 2023 amd64
- Amazon Linux 2023 aarch64
- Debian 11 aarch64
- Debian 12 amd64
- Debian 12 aarch64
- macOS 14 ARM
- macOS 14 x86_64
- FIPS 140-2 compliant Red Hat Enterprise Linux (RHEL) 9 x86_64

Client tools platforms added

Support has been added for PE client tools on the following operating system platforms:

- Amazon Linux 2023 amd64
- macOS 14 ARM

Solaris 11 packages now verified with GPG

Starting with PE 2021.7.8 and 2023.7, Solaris 11 agent packages are no longer signed with a DigiCert code signing certificate. Instead, you can verify the package's authenticity by using GPG-based verification with the provided .asc file.

Resolved issues

Replica promotion no longer corrupts file sync when lockless code deployment is enabled

In PE versions 2021.7.2 through 2021.7.7, and 2023.0 through 2023.6, if the lockless code deployment feature was enabled, using the disaster recovery workflow to promote a replica could lead to file sync corruption and code deployment failures. The issue is resolved in PE 2021.7.8 and 2023.7.

Fixed issue affecting recover_configuration cron job

In PE versions 2021.7.7 and 2023.6, the recover_configuration cron job could sometimes cause a Puppet Server restart, which in turn could cause an in-process provisioning of a replica to fail. The issue is resolved in PE 2021.7.8 and 2023.7.

Node-pinning issue fixed

In earlier versions of the Puppet Enterprise console, when a node group was set to match any rule, pinning a node resulted in the pinned node rule being incorrectly displayed in the main rules section rather than in the pinned nodes section. This issue is resolved in PE 2021.7.8.

Backup and restore commands automatically use Puppet binary path

In 2021.7.7, the puppet backup create and puppet backup restore commands would fail if the PATH variable didn't include the directory with the Puppet binary. This could occur, for example, when running the backup command from a cron job. Now, the full path to the Puppet binary is used automatically by the puppet backup create and puppet backup restore commands.

Security fixes

Addressed the following CVEs:

- CVE-2024-22871
- CVE-2024-1597
- CVE-2024-25710
- CVE-2024-26308
- CVE-2023-42503
- CVE-2024-46218

PE 2021.7.7

Released February 2024

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the leading-edge PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.y documentation](#).

Enhancements

Upgraded logback

To address CVE-2023-6378, logback is upgraded to version 1.3.14. If you want to use a customized setting for the logappender variable, see [Upgrade cautions](#) for information about avoiding disruptions in logging.

Platform support

Added agent platforms

Support is added for the following operating system platforms:

- AIX 7.3

Resolved issues

Upgraded concurrent-ruby to resolve issue that could cause Puppet Server memory leak

A known issue in the concurrent-ruby version packaged with PE 2021.7.5 and 2021.7.6 could cause Puppet Server memory leaks, resulting in gradual degradation of Puppet Server performance until the service crashed or was restarted. To resolve this issue, concurrent-ruby is updated to version 1.2.2.

Restoring PE from a backup no longer fails when puppet agent is running

Previously, when running `puppet-backup restore`, if a Puppet run was either already in progress or started during the restore process, the restore operation could fail with an error. This issue is fixed in PE 2021.7.7.

Restoring PE from a backup no longer fails if lockless code deployments are enabled

In previous PE versions, running `puppet-backup restore` resulted in a fatal error if the `puppet_enterprise::profile::master::versioned_deploys` parameter was set to `true`. The issue is fixed in PE 2021.7.7.

Setting the classifier_host parameter no longer causes failure in puppet-backup restore process

In previous versions, the `puppet-backup restore` process could fail in cases where the `puppet_enterprise::profile::master::classifier_host` parameter was defined. The issue is fixed in PE 2021.7.7.

Security fixes

Addressed the following CVEs:

- CVE-2023-6378
- CVE-2023-40167
- CVE-2023-36479
- CVE-2023-41900
- CVE-2023-5869
- CVE-2024-20952
- CVE-2024-20918
- CVE-2023-44487
- CVE-2023-5072
- CVE-2024-20932
- CVE-2023-38546

PE 2021.7.6

Released November 2023

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the STS PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.y documentation](#).

Enhancements

Updated common PQL queries in console

When configuring Puppet runs in the console, you can choose from a range of common Puppet Query Language (PQL) queries to target nodes for jobs and tasks. Because legacy facts are deprecated in Puppet 7, common queries that used legacy facts have been updated to use equivalent structured facts.

Platform support

Added agent platforms

Support is added for the following operating system platforms:

- Red Hat Enterprise Linux (RHEL) 9 ARM64
- Ubuntu 22.04 ARM64

Resolved issues

Installing packages with Ubuntu's Advanced Packaging Tool (APT) no longer causes restarts of pe-puppetserver and pe-orchestration-services

On Ubuntu 22.04, if you use the apt or apt-get commands to install new packages, the needrestart app no longer triggers unexpected restarts of pe-puppetserver and pe-orchestration-services.

Security fixes

Addressed the following CVEs:

- CVE-2023-40175
- CVE-2023-38545
- CVE-2023-36478
- CVE-2023-44487
- CVE-2023-4759
- CVE-2023-30589
- CVE-2023-5309

PE 2021.7.5

Released September 2023

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the STS PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.y documentation](#).

Enhancements

Classifier service flags unmappable legacy facts in node group rules

Legacy facts are deprecated in Puppet 7, which is the Puppet version included in this release, and are removed in Puppet 8, which is the Puppet version introduced in PE 2023.4. To support the transition away from legacy facts to structured facts, the classifier service in PE 2021.7.5 analyzes your node group rules and generates warning messages in the logs to flag uses of certain legacy facts that do not map to equivalent structured facts in Puppet 8. For more information about the removal of deprecated legacy facts in Puppet 8, see [Puppet upgrade in 2023.4](#).

Orchestrator HTTP-client limits can be configured to match infrastructure requirements

You can now specify HTTP-client connection limit parameters in the `puppet_enterprise::profile::orchestrator` class. You can set connection limits for authenticated and unauthenticated clients by specifying an integer value for the following parameters:

- `max_connections_per_route_authenticated`

- max_connections_total_authenticated
- max_connections_per_route_unauthenticated
- max_connections_total_unauthenticated

Orchestrator socket timeout is configurable

By default, whenever no data is available on the socket, the orchestrator waits for a maximum of 120,000 milliseconds before closing the HTTP connection. Now you can specify the maximum time before socket timeout by changing the default value of the `socket_timeout` parameter in the `puppet_enterprise::profile::orchestrator` class.

Improvements to error logging for the `puppet backup` command

Previously, error messages returned by the `puppet backup` command were generic in many cases. Now, descriptive error messages are displayed both in the terminal and in the log file, and you can use a `--debug` flag with `puppet backup` to extend error logging to all underlying Puppet commands.

Platform support

PE 2021.7.5 adds support for the following operating system platforms.

Added primary server platforms

Red Hat Enterprise Linux (RHEL) 9 x86_64

Ubuntu 22.04 amd64

Added agent platforms

macOS 13 ARM and x86_64

Added client tools platform

macOS 13 ARM and x86_64

Added patch management platforms

Red Hat Enterprise Linux (RHEL) 9 x86_64

With this release, support was removed for several previously deprecated platforms. Before upgrading to PE 2021.7.5, review the following list of removed platforms and the important information about removed platforms in [Platforms removed in 2021.0 and later](#) on page 181.

Removed agent platforms

CentOS 7 aarch64

macOS 10.15

Oracle Linux 7 aarch64

Red Hat 7 aarch64

Scientific Linux 7 aarch64

Removed client tool platforms

macOS 10.15

Deprecations and removals

Removed platforms

For information about platforms removed in this release, see the **Platform support** section.

Resolved issues

Installing Windows agent through the console no longer fails when option to test connections is selected

In PE 2021.7.2 and later, when installing Windows agents in the console's **Install agent on nodes** screen, checking the **Test Connections** checkbox before clicking **Add nodes** caused the process to hang indefinitely. The issue is resolved in PE 2021.7.5.

PE 2021.7.4

Released June 2023

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the STS PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.0 documentation](#).

Resolved issues

Security fix

Addressed CVE-2023-2530

PE 2021.7.3

Released May 2023

Important: PE 2021.7 is the current PE LTS series, and PE 2023 is the STS PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes), see [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023, see [Upgrading Puppet Enterprise in the 2023.0 documentation](#).

Enhancements

Improved performance when querying PuppetDB

This enhancement helps to improve performance for PuppetDB queries that contain large arrays, for example, if many nodes are enumerated or many terms are joined by a single "and" or "or" element.

Improved performance for the `each`, `map`, and `filter` functions in the Puppet language

Previously, the Puppet language built-in functions `each`, `map`, and `filter` showed poor performance and consumed unnecessary resources when run on JRuby software. The issue was resolved to enhance performance.

Puppet Server provides more reliable warnings when it cannot check for an update

By default, Puppet Server periodically checks whether a new version of Puppet Server is available. Previously, if Puppet Server could not connect to the update server, users were not provided with adequate information about the error. Starting with Puppet Server 7.10.1, a warning about the error is available in the log file.

Deprecations and removals

Deprecated PSON

In previous releases, Pure JavaScript Open Notation (PSON) was used in Puppet to serialize data for transmission.

PSON is deprecated in Puppet 7 and will be removed in Puppet 8.

Resolved issues

Tasks page is available following a software update

After upgrading PE from 2019.8 to 2021.7.1, the **Tasks** overview page in the PE console sometimes failed to load because of a timeout error. The issue is fixed in PE 2021.7.3 and 2023.1.

Enabling the lockless code deploy feature no longer causes performance issues in PuppetDB catalog compilation

When the `versioned_deploys` setting is enabled, Puppet previously reported the full directory path to the environment after resolving symbolic links as the source for resources in a catalog. Puppet now reports the path to the resource before resolving symbolic links in the environment path to help prevent instability of the PuppetDB instance.

Performance issue with Puppet agent runtimes is resolved

After an upgrade from PE 2019.8.12 to PE 2021.7.1, some users saw a significant increase in Puppet agent runtimes. The increase was caused by Facter 4, which was not using cached information to resolve facts. As a result, facts were resolved multiple times. The issue is now resolved to normalize the performance of the Puppet agent.

Certificates and keys can be backed up and restored by specifying the `certs` scope

Previously, if you ran the `puppet-backup create` command and specified a scope of `certs`, the command failed to back up the certificate authority root key and certificates. This issue occurred because Puppet 7 introduced a new default path for the certificate authority (CA) directory (`/etc/puppetlabs/puppetserver/ca`), but the `puppet-backup create` command failed to locate the new directory. Similarly, if you ran the `puppet-backup restore` command with a scope of `certs`, the restore operation failed. The CA directory issue is resolved so that backup and restore operations can run successfully.

Updates implemented to help users enter valid URLs

In previous versions of PE, the role-based access control (RBAC) service permitted the entry of invalid URLs when users specified the **Organizational URL** setting. Login attempts would then fail with the following error message:

```
'Invalid settings: organization_not_enough_data'
```

In PE 2021.7.3 and 2023.1, the RBAC service is updated to enforce valid URLs when users create or update a connection to a Security Assertion Markup Language (SAML) identity provider, and the PE console displays a warning if the user enters an invalid URL for the **Organizational URL** setting.

Timeouts can be specified for SAML authentication

Previously, when users configured the PE console to specify `session-timeout` and `session-maximum-lifetime` values, the settings were applied to Lightweight Directory Access Protocol (LDAP) tokens and local login tokens. However, the specified settings were not applied to SAML tokens, which are used for authentication with a SAML identity provider. The issue is corrected to ensure that the specified settings also apply to SAML session lifetimes.

User-defined temporary directory is honored during PE restore operations

After you back up your PE infrastructure, you can use the `puppet-backup restore` command to restore the backup. Previously, if you set the `-tmpdir` flag or the `TMPDIR` environment variable to specify a temporary directory for restore operations, the directory was not honored, and the default `/tmp` directory was used in some cases. In addition, some files were not cleaned up after the restore operation. This issue is corrected to ensure that the user-specified directory is used and all temporary files are removed after the restore operation.

Issue that caused an unexpected increase in CPU usage is resolved

In PE 2021.7.1, 2021.7.2, and 2023.0, an issue with Puppet Server caused an unexpected increase in central processing unit (CPU) usage in some environments. CPU usage continued to grow and some operations took

longer than expected until the Puppet Server service was restarted. This issue is resolved in PE 2023.1 and 2021.7.3.

Security fixes

Addressed CVE-2023-1894 and CVE-2023-26048.

PE 2021.7.2

Released January 2023

Important: PE 2021.7 is the current PE LTS series, and PE 2023.0 is the new STS PE series.

For information about upgrading from 2019.8.z to 2021.7 (and earlier 2021.y series release notes) go to [What's new since PE 2019.8](#) on page 55 and [Upgrading Puppet Enterprise](#) on page 182.

For information about upgrading to 2023.0, go to [Upgrading Puppet Enterprise in the 2023.0 documentation](#).

Enhancements

`recover_configuration` command recreates `nodes` files

Previously, the `puppet infrastructure recover_configuration` command merged new values into the `nodes` files (at `/etc/puppetlabs/enterprise/conf.d/nodes`) instead of overwriting the files. This process caused problems if you deleted a value relevant to one or more nodes, because the deleted value would remain in these files and continue to be applied.

Now, the `recover_configuration` command fully rewrites the `nodes` files on each invocation. This process matches how the command handles changes to the `user_data.conf` file.

Improved performance when regenerating agent certificates for multiple agents

The `puppet infrastructure run regenerate_agent_certificate` action is now faster when you [Regenerate agent certificates](#) on page 840 for multiple agents. You can also now use the `agent_pdb_query` parameter to run a PDB query to generate a list of agents for which you want to regenerate certificates.

This action now uses the Puppet Server CA API endpoints directly, rather than relying on the `puppetserver ca` CLI, as it did previously. This process is faster, but, if you encounter problems, you can revert to the previous behavior by including `use_puppetserver_cli=true` in the command.

Specify Code Manager worker cache cleanup interval

The `deploy-pool-cleanup-interval` parameter specifies how often workers pause to clean their on-disk caches. Learn more about this setting in [Code Manager parameters](#) on page 791.

Platform support

This version adds support for the following platforms:

Agent platforms

Solaris 10 (SPARC, i386)

Client tools platforms

Solaris 10 (SPARC, i386)

Resolved issues

Code Manager respects `full_deploy` setting in Hiera

The `full_deploy` parameter is now correctly applied when you [Customize Code Manager configuration in Hiera](#) on page 785.

Previously, the `full_deploy` parameter value was disregarded when included in a Code Manager configuration in Hiera. As a work-around, you could create a separate `.conf` file to manually manage this parameter.

Important: If you created a `.conf` file for the `full_deploy` parameter, you must remove this file and reconfigure the parameter in Hiera (as described in [Configuring module deployment scope](#) on page 787).

Certain plans correctly restore puppet service to pre-plan state

Due to a bug introduced in PE 2021.6, some plans that must stop the `puppet` service while the plans run were not restoring the `puppet` service to its pre-plan state after the plan finished running.

The four affected plans, and their associated `puppet infra` commands, are as follows:

- The `secondary_cert_regen` plan, which is triggered by `puppet infra run regenerate_compiler_certificate` and `puppet infra run regenerate_replica_certificate`
- The `convert_legacy_compiler` plan, which is triggered by `puppet infra run convert_legacy_compiler`
- The `reprovision_replica` plan, which is specifically triggered by `puppet infra upgrade replica --only-recreate-databases`
- The `enable_ha_failover` plan, which is triggered by `puppet infra run enable_ha_failover`

Important: If you ran any of these four plans since upgrading to PE 2021.6, check the state of the `puppet` service on your infrastructure nodes.

PuppetDB database user can purge reports

An issue was fixed to help ensure that the PuppetDB database user can purge reports.

Corrected fact list handling in some PE console UI components

Some user interface (UI) components in the PE console use fact lists. A recent change caused these components to use the entire list of fact names. This process caused performance problems in environments with many facts. The handling of fact lists was updated to fix this issue and improve performance.

Orchestrator code directories excluded from `puppet-backup create --scope=config`

When [Customize scope of backup and restore](#) on page 847, the orchestrator code directories (specifically `opt/puppetlabs/server/data/orchestration-services/data-dir` and `opt/puppetlabs/server/data/orchestration-services/code`) are excluded when you specify the `config` scope.

These directories are included in the `code` scope.

Garbage collection log fixes

The introduction of Java 11 resulted in two issues pertaining to garbage collection logs. The issues are now fixed:

Dates and times are included in garbage collection logs.

The maximum volume of retained garbage collection logs is 256 MB.

Security fixes

Addressed CVE-2022-41946 and CVE-2022-41404.

PE 2021.7.1

Released October 2022

Important: PE 2021.7 is our new PE LTS series. If you're preparing to upgrade, or looking for earlier 2021.y release notes, go to [What's new since PE 2019.8](#) on page 55.

For those awaiting the new STS, we're still getting things ready for the first release in that series. We thank you for your patience.

New features

Stop in-progress plans

Use [POST /command/stop_plan](#) on page 686 to stop an orchestrator plan job that is currently running.

Platform support

Deprecated and removed platforms are listed under **Deprecations and removals**.

This version adds support for these platforms:

Agent platforms

Fedora 36

Patch management platforms

Fedora 36

Deprecations and removals

Deprecated agent platforms

Debian 9

Fedora 32, 34

Deprecated patch management platforms

Debian 9

Fedora 34

Resolved issues

Deactivated scheduled jobs could still run.

If you deactivate a recurring scheduled job, the inactive job no longer continues to run after restarting pe-orchestration-services.

Deactivated jobs aren't visible in the console or in responses from the [Scheduled jobs endpoints](#) on page 717, but, prior to this fix, some deactivated jobs could run again at their next scheduled interval as if they had not been deactivated. This issue only impacted deactivated scheduled jobs that had run within the `job_prune_threshold` limit *after* restarting pe-orchestration-services.

Orchestrator didn't properly periodically prune jobs

Fixed a calculation error introduced in PE 2021.5 that caused job records to be stored beyond the `job_prune_threshold` limit.

`regenerate_agent_certificate` couldn't verify node type if client tools were installed through a package resource

When you run the `puppet infra run regenerate_agent_certificate` command, the plan can now verify that a node isn't an infrastructure node if the `pe-client-tools` package was installed on the node through a package resource.

RBAC API `command/config/remove-disclaimer` endpoint erroneously required Content-Type header

The [POST /command/config/remove-disclaimer](#) on page 358 endpoint no longer requires a Content-Type header, because requests to this endpoint have no body content.

Internal task jobs shared primary task thread pool

Internal task jobs (such as tasks that force stop other tasks) no longer run on the same thread pool as your user-initiated tasks. This allows internal tasks to queue separately from other tasks. For example, requests to [POST /command/stop](#) on page 685 don't get stuck waiting if the regular task queue is full.

Improved PuppetDB disaster recovery sync performance

The PuppetDB disaster recovery sync process transferred more reports than necessary when syncing reports, which sometimes caused timeouts.

Empty task metadata files prevented you from running tasks in the console

Loading empty task metadata files no longer cause errors.

Some `puppet infrastructure` commands failed when restarting the `puppet` service

Previously, several `puppet infrastructure` commands failed when restarting the `puppet` service at the end of the action. While the service had successfully restarted, the effected actions couldn't properly detect the restart, which caused them to fail. This has been fixed.

PE 2021.7.0

Released August 2022

Important: PE 2021.7 is our new PE LTS. Expect to see changes to the documentation ahead of our next STS release. We apologize for any inconvenience.

If you're preparing to upgrade or looking for earlier 2021.y release notes, go to [What's new since PE 2019.8](#) on page 55.

New features

Force stop in-progress Puppet runs

By default, [POST /command/stop](#) on page 685 prevents new runs from starting, but allows in-progress runs to finish. Now you can use the `force` option to block new runs *and* stop in-progress runs. This is useful, for example, if you need to stop a task that is hanging.

`pe_status_check` module bundled with PE

The `pe_status_check` module helps keep your PE installation in an ideal state. Read [About the pe_status_check module](#) on page 400 to learn how the module works and how to get the module's reports.

Important: If you have previously specified a version of this module, from the Forge or other sources, in your code, we recommend removing this version before upgrading to allow the version bundled with PE to be asserted.

New Orchestrator scheduling API

This release includes a new scheduling API for the orchestrator, which introduces several new `scheduled_jobs` endpoints and deprecates the previous scheduling API's endpoints (for a list of deprecated endpoints, see **Deprecations and removals** for this release, below).

Existing scheduled jobs are automatically migrated to the new scheduling system, and the PE console now uses the new API (but there is no change to the UI).

With the new API, you can edit scheduled jobs; however, this functionality is currently only available through the API (not yet available in the PE console). To learn more about the new endpoints, go to [Scheduled jobs endpoints](#) on page 717.

Tools that rely on the deprecated endpoints must be upgraded to use the new endpoints.

Use the RBAC API to set the disclaimer text on the console login page

You can use the RBAC API v1 [Disclaimer endpoints](#) on page 357 to configure the disclaimer text that appears on the PE console login page.

Automatically sync LDAP user details and group membership

Prior to this release, user details and group membership for LDAP-based users only refreshed when users logged in. Now, LDAP group bindings, user names, and descriptions update automatically every 30 minutes (by default) for every LDAP user in the system. If a user is no longer present in LDAP or has no group bindings, all user-group associations are removed from the user and all of the user's known tokens are revoked.

You can disable automatic refresh or change the refresh time by changing the `puppet_enterprise::profile::console::ldap_sync_period` parameter. Learn more about this parameter in [Configure RBAC and token-based authentication settings](#) on page 224.

Stop LDAP users from logging in if they have no group membership

You can use the `exclude_groupless_ldap_users` setting to prevent LDAP users with no group memberships from logging in. This setting is off by default. To learn how to enable this setting, go to [Require LDAP group membership to log in](#) on page 267.

Metrics API v2 documentation

The [Metrics API v2](#) on page 411 uses the Jolokia library to query Orchestrator service metrics. This version of the API has been available for some time, but it was only described in the open source Puppet documentation.

Disaster recovery support for FIPS platforms

[Disaster recovery](#) on page 248 is now supported for FIPS 140-2 compliant Red Hat Enterprise Linux (RHEL) 7 and 8.

Enhancements

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

[Orchestrator API v1](#) on page 678 endpoints that return pagination containing the total number of jobs (such as [GET /jobs](#) on page 703, [GET /scheduled_jobs \(deprecated\)](#) on page 731, and [GET /plan_jobs](#) on page 736) now return "total": 0, instead of "total": null, when there are no jobs.

Activity service API /v2/events endpoint returns more information for orchestrator events

Responses from [GET /v2/events](#) on page 376 containing information about orchestrator events (Puppet agent runs and task runs) now report additional information about the job start time, end time, duration, and status.

Upgraded JRuby

We are now shipping JRuby 9.3.4.0.

Addressed CVEs

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

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

Platform support

Ubuntu 16.04 is no longer a supported agent platform.

This version adds support for these platforms:

Agent

macOS 12 M1

Ubuntu (General Availability kernels) 22.04 x86_64

Microsoft Windows 11 x64

Client tools

Ubuntu (General Availability kernels) 22.04 x86_64

macOS 12 M1, M2

Patch management

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

Deprecations and removals

Ubuntu 16.04 is no longer a supported agent platform.

The following endpoints are deprecated due to the release of several new [Scheduled jobs endpoints](#) on page 717 for the Orchestrator API. Tools that rely on deprecated endpoints must be upgraded to use the new endpoints. Existing scheduled jobs are automatically migrated to the new scheduling system that uses the new endpoints.

[GET /scheduled_jobs \(deprecated\)](#) on page 731

Replaced by [GET /scheduled_jobs/environment_jobs](#) on page 717 and [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721

[DELETE /scheduled_jobs/<job-id> \(deprecated\)](#) on page 731

Replaced by [PUT /scheduled_jobs/environment_jobs/<job-id>](#) on page 730

[POST /command/schedule_deploy \(deprecated\)](#) on page 698

Replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

[POST /command/schedule_plan \(deprecated\)](#) on page 699

Replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

[POST /command/schedule_task \(deprecated\)](#) on page 698

Replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

Resolved issues

full-deploy didn't override --incremental

Code Manager's full-deploy option, used for [Configuring module deployment scope](#) on page 787, now correctly overrides the default --incremental deploy behavior.

Code Manager couldn't fetch code on FIPS platforms

On FIPS platforms running PE versions 2021.5 or 2021.6, Code Manager and r10k couldn't fetch code from your code repo due to libssh attempting to use algorithms that are not allowed on FIPS. In PE 2021.7, the disallowed algorithms are disabled in libssh, allowing Code Manager and r10k to successfully fetch code.

An unreachable replica consumed all of the primary server's disk space

Previously, if a provisioned replica became unreachable, the associated primary server could quickly run out of disk space, causing a complete interruption to PE services. In larger installations, an outage could occur in under an hour. Excessive disk usage was caused by the PE-PostgreSQL service on the primary server retaining change logs that the replica hadn't acknowledged.

To resolve this, we limited available disk space for the pg_wal directory. To learn more and tune this setting in your installation, refer to [PostgreSQL WAL disk space](#) on page 210.

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

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

Some RBAC endpoints returned an incorrect Content-Type

Responses for the following endpoints now return the correct Content-Type: [POST /users/<uuid>/password/reset](#) on page 353, [POST /auth/reset](#) on page 354, and [PUT /users/current/password](#) on page 355.

LDAP with anonymous binding sometimes prevented Console Services from starting or restarting

Previously, if you use anonymous binding, or another configuration with a zero-length password, Console Services sometimes couldn't start or restart. This could cause upgrade failures when upgrading to PE version 2021.4 through 2021.6 from a version earlier than 2021.4. This is resolved.

Orchestrator doesn't restart unexpectedly during the convert_legacy_compiler plan

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

Some SSO configuration fields weren't marked as required

The **Organization** and **Contacts** fields on the **SSO Configuration** page are now correctly marked as required.

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

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

Incorrect run-time for splayed agent runs

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

Sensitive parameters sometimes exposed in cleartext in job results

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

PE known issues

These are the known issues in PE 2021.7.

Installation and upgrade known issues

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

Puppet Server JVM segfaults during RHEL 7 to RHEL 8 migration

During migration of PE from Red Hat Enterprise Linux (RHEL) 7 to RHEL 8, if your `puppetlabs-stdlib` module version is older than 9.1.0, the puppetserver Java Virtual Machine (JVM) might experience segfaults at seemingly random intervals. This issue is caused by underlying logic related to password hashing. To avoid this problem, upgrade the `puppetlabs-stdlib` module to version 9.1.0 or newer.

Converting legacy compilers fails with an external certificate authority

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

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

To work around this issue when it appears:

1. Log on to the CA server and manually sign certificates for the compiler.
2. On the compiler, run Puppet: `puppet agent -t`
3. Unpin the compiler from **PE Master** group, either from the console, or from the CLI using the command: `/opt/puppetlabs/bin/puppet resource pe_node_group "PE Master" unpinned="<COMPILER_FQDN>"`

4. On your primary server, in the `pe.conf` file, remove the entry
`puppet_enterprise::profile::database::private_temp_puppetdb_host`
5. If you have an external PE-PostgreSQL node, run Puppet on that node: `puppet agent -t`
6. Run Puppet on your primary server: `puppet agent -t`
7. Run Puppet on all compilers: `puppet agent -t`

Converted compilers can slow PuppetDB in multi-region installations

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

Installing and upgrading error on Red Hat Enterprise Linux 9

Installing or upgrading to 2021.7.10 on Red Hat Enterprise Linux 9 results in an error and failed installation. As a workaround:

1. Force upgrade both packages:

```
rpm -U --force ./pe-r10k-2021.7.10.0-1.el9.x86_64.rpm
rpm -U --force ./pe-bolt-server-2021.7.10.0-1.el9.x86_64.rpm
```

2. Re-run the installer.

Disaster recovery known issues

There are no known issues for disaster recovery in this release.

FIPS known issues

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

FIPS-enabled PE 2021.7 and later can't use the default system cert store

PE 2021.7 and later FIPS builds can't use the default system cert store, which is used automatically with some reporting services. This setting is configured by the `report_include_system_store` Puppet parameter that ships with PE.

Removing the `puppetcacerts` file (located at `/opt/puppetlabs/puppet/ssl/puppetcacerts`) can allow a report processor that eagerly loads the system store to continue with a warning that the file is missing.

If HTTP clients require external certs, we recommend using a custom cert store containing only the necessary certs. You can create this cert store by concatenating existing `pem` files and configuring the `ssl_trust_store` Puppet parameter to point to the new cert store.

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 OpenSSL library, such as `hiera-eyaml`. As a workaround, you can use a non-FIPS-enabled primary server with FIPS-enabled agents, which limits the issue to situations where only the primary uses the Ruby library. This limitation does not apply to versions 1.1.0 and later of the `splunk_hec` module, which supports FIPS-enabled servers. The [FIPS Mode](#) section of the module's Forge page explains the limitations of running this module in a FIPS environment.

Configuration and maintenance known issues

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

SAML Organization URL accepts invalid values

When you [Connect to a SAML identity provider](#) on page 290, make sure the **Organization URL** is a valid, well-formed URL. Supplying an invalid value does not produce a field validation error, but it does cause the following error at login: `Invalid settings: organization_not_enough_data`.

`puppet infrastructure tune` fails with multi-environment environmentpath

The [puppet infrastructure tune command](#) on page 205 fails if `environmentpath` (in your `puppet.conf` file) is set to multiple environments. To avoid the failure, comment out this setting before running this command. For details about the `environmentpath` setting, refer to [environmentpath in the open source Puppet documentation](#).

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 this warning in the command-line output or logs: `Illegal reflective access operation ... All illegal access operations will be denied in a future release`

These warnings are internal to PE service components and have no impact on their functionality. You can safely disregard them.

Orchestration services known issues

There are no known issues related to orchestration services at this time.

Console known issues

There are no known issues related to the console and console services at this time.

Patching known issues

These are the known issues for patching in this release.

Patching fails with excluded YUM packages

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

Code management known issues

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

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.

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.

The toggle_lockless_deploys plan does not configure the replica in disaster recovery architecture

The `toggle_lockless_deploys` plan does not properly switch over a replica in disaster recovery to have lockless deploys because doing so requires updating Hiera data. Users who already have disaster recovery enabled and are toggling lockless deploys are recommended to update their `pe.conf` after running the plan.

The toggle lockless deploys plan runs some actions verbosely, and failures are expected while polling for changes

While polling for changes, do not prematurely abort the `toggle_lockless_deploys` plan run if you see errors on the console. Wait until the plan fully completes before assessing whether the failures were an expected part of polling for changes or not.

The toggle lockless deploys plan does not support Ubuntu 18.04

The `toggle_lockless_deploys` plan currently supports Ubuntu 20.04 and 22.04. The plan supports Ubuntu 18.04 in the next release.

Code deployment fails from AzureDevOps with a rsa-sha error in Puppet Enterprise®

Deploying code via r10k or Code Manager from Azure DevOps (ADO) with a `rsa-sha2` key fails with a deprecated `rsa-sha` error. To fix the issue, change your authentication method to HTTPS and complete the steps outlined here: [Knowledge base article](#).

What's new since PE 2019.8

This page describes the new features, enhancements, deprecations, and other notable changes since the previous LTS release (2019.8), specifically PE versions 2021.0 through 2021.7.0. The previous LTS release stream comprised PE versions 2019.8.0 through 2019.8.12.

This page does not include resolved issues because most bug fixes were applied to both the 2019.8.z and 2021.y streams at the time of resolution, except those that only impacted one stream or the other. For bug fixes included in 2021.7.0, refer to the [2021.7.0 release notes](#). For information about outstanding, unresolved issues in 2021.7.z, refer to the [PE known issues](#) on page 52.

Some, but not all, features and changes described on this page applied to both the 2019.8.z and 2021.y streams. However, this page does not specify the interim release number for each feature or change. You can find the original release notes for each interim release (and release notes for the 2019.8.z series) in the [Documentation for other PE versions](#) on page 31. Furthermore, this page stops at 2021.7.0; to learn about resolved issues, new features, deprecations, and other changes in later releases, refer to the [PE release notes](#) on page 36.

Important: Before upgrading to 2021.7:

- Review the [Upgrade cautions](#) on page 180 and [Upgrade paths](#) on page 179 for important information that could impact your upgrade.
- Get familiar with the latest [System requirements](#) on page 96 including hardware requirements, supported operating systems, supported browsers, and network configurations.

Highlights

PostgreSQL upgrade

PE version 2021.6 upgraded PostgreSQL to version 14. When you upgrade to 2021.7, your PostgreSQL instance is migrated from version 11 to version 14.



CAUTION: Review information about the [PostgreSQL 14 upgrade in PE 2021.6](#) on page 180 before upgrading.

If PE does not manage your PostgreSQL instance, you must [Upgrade your unmanaged PostgreSQL installation](#) before upgrading your primary server, compilers, and agents to 2021.7.

Lockless code deploys are no longer experimental

Since debuting as an experimental feature, we've worked with customers to enable [Lockless code deploys](#) on page 783 over previous releases. The feature has been stable since the last bug fix in version 2021.2, and we're confident lockless code deploys are ready for prime time.

Run plans without blocking code deployments

You can allow orchestrator to run plans without blocking your code deployments, and you can deploy code without waiting for plans to finish. For instructions, refer to [Running plans alongside code deployments](#) on page 651.

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

Force stop in-progress Puppet runs

By default, [POST /command/stop](#) on page 685 prevents new runs from starting, but allows in-progress runs to finish. Now you can use the `force` option to block new runs *and* stop in-progress runs. This is useful, for example, if you need to stop a task that is hanging.

Automatically sync LDAP user details and group membership

Prior to 2021.7, user details and group membership for LDAP-based users only refreshed when users logged in. Now, LDAP group bindings, user names, and descriptions update automatically every 30 minutes (by default) for every LDAP user in the system. If a user is no longer present in LDAP or has no group bindings, all user-group associations are removed from the user and all of the user's known tokens are revoked.

You can disable automatic refresh or change the refresh time by changing the `puppet_enterprise::profile::console::ldap_sync_period` parameter. Learn more about this parameter in [Configure RBAC and token-based authentication settings](#) on page 224.

Stop LDAP users from logging in if they have no group membership

You can use the `exclude_groupless_ldap_users` setting to prevent LDAP users with no group memberships from logging in. This setting is off by default. To learn how to enable this setting, go to [Require LDAP group membership to log in](#) on page 267.

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. Go to [SAML authentication](#) on page 288 to learn about configuring SAML connections in PE.

Add a custom disclaimer banner to the console

You can [Create a custom login disclaimer](#) on page 267 for your PE console login page.

Disaster recovery support for FIPS platforms

[Disaster recovery](#) on page 248 is now supported for FIPS 140-2 compliant Red Hat Enterprise Linux (RHEL) 7 and 8.

API changes

New Orchestrator scheduling API endpoints

These new endpoints replaced five other scheduling endpoints (described below in **Deprecated endpoints**). Existing scheduled jobs are automatically migrated to the new scheduling system.

- [GET /scheduled_jobs/environment_jobs](#) on page 717
- [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721
- [POST /scheduled_jobs/environment_jobs](#) on page 724
- [PUT /scheduled_jobs/environment_jobs/<job-id>](#) on page 730

New RBAC API endpoints

- [Disclaimer endpoints](#) on page 357
- [GET /v2/users](#)
- [POST /command/roles/add-users](#) on page 334
- [POST /command/roles/remove-users](#) on page 335
- [POST /command/roles/add-groups](#) on page 335
- [POST /command/roles/remove-groups](#) on page 336
- [POST /command/roles/add-permissions](#) on page 336
- [POST /command/roles/remove-permissions](#) on page 337
- [POST /command/users/revoke](#) on page 325
- [POST /command/users/reinstate](#) on page 325
- [POST /command/users/add-roles](#) on page 324
- [POST /command/users/remove-roles](#) on page 325

Use Puppet Server API to update CRLs

Supply a list of CRL PEMs to the [certificate_revocation_list](#) endpoint to insert updated copies of the applicable CRLs into the trust chain. The CA updates matching CRLs saved on disk if the submitted ones have a higher CRL number than their counterparts. Use these endpoints if your CRLs require frequent updates. Do not use these endpoints to update the CRL associated with the Puppet CA signing certificate (only earlier ones in the certificate chain).

Request changes

Several endpoints have additional keys and/or values that you can use in your requests. Visit the page for each endpoint to learn about these additions.

- [GET /jobs](#) on page 703 requests allow `min_finish_timestamp` and `max_finish_timestamp`.
- [GET /plan_jobs](#) on page 736 requests allow `min_finish_timestamp`, `max_finish_timestamp`, `order`, and `order_by`.
- [GET /jobs/<job-id>/nodes](#) on page 710 requests allow `state`, `order` and `order_by`.
- [GET /v1/events](#) on page 372 and [GET /v2/events](#) on page 376 requests allow `order`.
- [GET /usage](#) on page 753 requests allow `events`.
- [POST /command/environment_plan_run](#) on page 696 requests allow `type` (within the `params` object) and `userdata`.
- [POST /command/deploy](#) on page 681, [POST /command/task](#) on page 687, and [POST /command/plan_run](#) on page 695 requests allow `userdata`.
- [POST /command/stop](#) on page 685 requests allow `force`, which blocks new runs *and* stop in-progress runs.

Response changes

Responses from several endpoints have additional keys. Visit the page for each endpoint to learn about these additions.

[GET /jobs](#) and [GET /plan_jobs](#) on page 736 responses include `userdata`, `duration`, `created_timestamp`, and `finished_timestamp`. Also, the pagination objects returned by these endpoints report `"total": 0`, instead of `"total": null`, when there are no jobs.

[GET /jobs/<job-id>](#) on page 708 and [GET /plan_jobs/<job-id>](#) on page 740 responses include `userdata`.

[GET /v2/events](#) on page 376 responses containing information about orchestrator events (Puppet agent runs and task runs) include additional information about the job start time, end time, duration, and status.

Deprecated endpoints

Important: Tools that rely on the deprecated endpoints must be upgraded to use the new endpoints.

LDAP GET `/v1/ds` endpoint was deprecated in favor of the more secure v2 [GET /ds](#) on page 367 endpoint.

[GET /scheduled_jobs \(deprecated\)](#) on page 731 replaced by [GET /scheduled_jobs/environment_jobs](#) on page 717 and [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721

[DELETE /scheduled_jobs/<job-id> \(deprecated\)](#) on page 731 replaced by [PUT /scheduled_jobs/environment_jobs/<job-id>](#) on page 730

[POST /command/schedule_deploy \(deprecated\)](#) on page 698 replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

[POST /command/schedule_plan \(deprecated\)](#) on page 699 replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

[POST /command/schedule_task \(deprecated\)](#) on page 698 replaced by [POST /scheduled_jobs/environment_jobs](#) on page 724

Bundled module changes

pe_status_check module included in PE

The `pe_status_check` module helps keep your PE installation in an ideal state. Read [About the pe_status_check module](#) on page 400 to learn how the module works and how to get the module's reports.

Important: If you have previously specified a version of this module, from the Forge or other sources, in your code, we recommend removing this version before upgrading to allow the version bundled with PE to be asserted.

Puppet metrics collector module included in PE

The Puppet metrics collector module collects Puppet metrics by default, but system metrics collection is disabled. To enable the module to collect system metrics, change `puppet_enterprise::enable_system_metrics_collection` to `true`.

Important: If you have already downloaded the metrics collector module from the Forge, you must either uninstall your copy of the module or upgrade it to the version installed with PE.

PE databases module included in PE

The `pe_databases` module is bundled with PE and enabled by default.

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

To disable this module, set `puppet_enterprise::enable_database_maintenance` to `false`.

Removed pe_java_ks module

The `pe_java_ks` module has been removed from PE packages. If you have any references to the packaged module in your code base, you must remove these references to avoid errors in catalog runs.

Certificate, access, and security-related changes

Upgraded Bouncy Castle

We are now shipping Bouncy Castle 1.70, which has improved support for TLSv1.3.

Updated PostgreSQL driver

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

Certificate, CA, CRL, and related changes

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

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

The `puppetserver ca prune` action runs during upgrades. On upgrade, the CA CRL is purged of duplicate entries, potentially making it a much smaller file. The Puppet CA also no longer adds duplicate entries to the CRL in the first place.

As part of the ongoing effort to remove harmful terminology, the command to regenerate primary server certificates has been renamed `puppet infrastructure run regenerate_primary_certificate`.

Use the `crl_refresh_interval` parameter to enable agents to re-download their CRLs on regular intervals.

The default CA directory has moved to `/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
```

Passwords and tokens

For improved security, the lookup password is no longer preserved when the LDAP configuration page is reloaded or revisited in the console. You must enter the lookup password every time you make a change to the LDAP configuration, and it is required if there is a lookup user specified.

You can switch the algorithm PE uses to store passwords from the default SHA-256 to argon2id by configuring new password algorithm parameters. To configure the algorithm, see [Configure the password algorithm](#) on page 227.

Note: Argon2id is not compatible with FIPS-enabled PE installations.

There are configurable [Password complexity parameters](#) on page 233 that local users see as requirements when creating a new password. For example, Usernames must be at least {8} characters long.

RBAC generates and accepts only cryptographic tokens, instead of JSON web tokens (jwt), for password resets.

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

Prevent replay attacks in SAML

SAML can now handle replay attacks by storing message IDs with their timestamps and rejecting message IDs that have been recently used, which prevents a bad actor from replaying a previously valid message to gain access. Stored message IDs are purged every 30 minutes.

Encrypt backups

Use the `puppet backup create` command with an optional `--gpgkey` to encrypt backups.

Return sensitive data from tasks

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

Use masked inputs for sensitive parameters

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

Automatically sync LDAP user details and group membership

LDAP group bindings, user names, and descriptions update automatically every 30 minutes (by default) for every LDAP user in the system. If a user is no longer present in LDAP or has no group bindings, all user-group associations are removed from the user and all of the user's known tokens are revoked. Learn more about this parameter in [Configure RBAC and token-based authentication settings](#) on page 224.

Stop LDAP users from logging in if they have no group membership

You can use the `exclude_groupless_ldap_users` setting to prevent LDAP users with no group memberships from logging in. This setting is off by default. To learn how to enable this setting, go to [Require LDAP group membership to log in](#) on page 267.

Code Manager, r10k, and file sync changes

File sync client status output

Profiling metrics are reported for versioned deploys and basic deploys in the file-sync client's debug status output.

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

File sync always overrides the contents of the live directory when syncing.

This corrects any local changes made in the live directory outside of Code Manager's workflow.

Configure module deployment scope

By default, Code Manager utilizes the r10k `--incremental` deploys feature for improved performance. Incremental deploys only sync modules whose definitions allow their version to "float" (such as Git branches) and modules whose definitions have been added or changed since the environment's last deployment. SVN modules are not supported. To disable this behavior (and deploy all module code regardless of change or float status), set Code Manager's `full_deploy` parameter to `true`, as described in [Configuring module deployment scope](#) on page 787.

Custom Forge server authentication

Code Manager now supports authentication to custom servers through the `authorization_token` in the `forge_settings` parameter when [Configuring Forge settings](#) on page 787 for Code Manager.

Include r10k stacktrace in failed deployment output

Use the `r10k_trace` parameter in your [Code Manager settings](#) on page 782 to include r10k stacktrace in the error output for failed deployments.

Performance improvements

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

Previously, Code Manager deployed whole modules to disk, often including the spec directory. The spec directory is only used for testing and is not useful in a production environment. Now, Code Manager deletes

the spec dirs from deployments to decrease disk size. You can disable this behavior for each module by setting `exclude_spec` to `false` for relevant module declarations in your Puppetfile.

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

The `environment_timeout` setting's default value is now 5m.

Removed settings

Removed `environment_timeout_mode`.

Replaced `purge-whitelist` with `purge-allowlist`. No backwards compatibility. You must update your Code Manager and file sync configurations to use `purge-allowlist`.

Patching changes

Run patches sequentially in the `group_patching` plan

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

Avoid spam during patching

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

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

The `post_patching_scriptpath` parameter in the `pe_patch` class allows you to run an executable script or binary on a target node after patching is complete.

The `pre_patching_command` parameter has been renamed to `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 are moved to directories that are less likely to be read-only.

If you use patch-management, be aware of the following:

- Before upgrading, you might want to 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.

Other changes

Upgraded JRuby

We are now shipping JRuby 9.3.4.0.

Optimized some PuppetDB queries

Improved performance of queries that `puppet infrastructure` uses to look up Puppet infrastructure node certnames.

Report compilation failure results for apply blocks

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

Run the `puppet infra run` command with WinRM

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

More options when running the support script

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

Export node data from task runs as CSV

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

Differentiate backup and restore logs

Backup and restore log files are now appended with timestamps, and they aren't overwritten with each backup or restore action.

Clean up old PE versions with smarter defaults

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

Customize value report estimates

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

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

Replica upgrades issue warnings instead of errors 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 (with `puppet infra provision replica --enable`), the command times out if there are issues syncing PuppetDB, and provides instructions for fixing any issues and separately provisioning the replica.

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.

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.

Configure how many times the orchestrator allows status request timeouts

Configure the `allowed_pcp_status_requests` parameter to define how many times an orchestrator job allows status requests to time out before the job fails.

Add custom parameters when installing agents in the console

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

Update facts cache terminus to use JSON or YAML

The facts cache terminus is now JSON by default. You can configure the `facts_cache_terminus` parameter to switch from JSON to YAML.

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.

Renamed settings

These settings were replaced to removed harmful terminology. No guaranteed backwards compatibility. You must update your configurations and code to the new settings as part of your upgrade to 2021.7.

```
master-conf-dir is now server-conf-dir
master-code-dir is now server-code-dir
master-var-dir is now server-var-dir
master-log-dir is now server-log-dir
master-run-dir is now server-run-dir
master_uris is now primary_uris
```

Platform support

PE 2021.0 through 2021.6 added support for these platforms:

Primary server platforms

- AlmaLinux x86_64 for Enterprise Linux 8
- Amazon Linux 2
- Red Hat Enterprise Linux 8 FIPS x86_64
- Rocky Linux x86_64 for Enterprise Linux 8
- SUSE Linux Enterprise Server 15 x86_64
- Ubuntu (General Availability kernels) 20.04 amd64

Agent platforms

- AlmaLinux x86_64 for Enterprise Linux 8
- Debian 11 (Bullseye) amd64
- Fedora 32, 34
- macOS 11 x86_64
- macOS 12 x86_64, M1
- Microsoft Windows 11 x64
- Microsoft Windows Server 2022 x86_64
- Red Hat Enterprise Linux 8 FIPS x86_64
- Red Hat Enterprise Linux 8 ppc64le
- Red Hat Enterprise Linux 9 x86_64
- Rocky Linux x86_64 for Enterprise Linux 8
- Ubuntu 18.04 aarch64
- Ubuntu 20.04 aarch64
- Ubuntu 22.04 x86_64

Client tools platforms

- macOS 11

macOS 12 M1, M2

Ubuntu 22.04 x86_64

Patch management platforms

Amazon Linux 2

Microsoft Windows 11 x64

Ubuntu 22.04 x86_64

For comprehensive platform support information, refer to:

- [Supported operating systems](#) on page 98
- [Supported PE client tools operating systems](#) on page 173
- [Patch management OS compatibility](#) on page 575

Platform deprecations and removals

Deprecated primary server platforms

CentOS 8

Deprecated agent platforms

CentOS 8

Debian 8

Enterprise Linux 5

Enterprise Linux 7 ppc64le

Fedora 30, 31

macOS 10.14

Microsoft Windows 7, 8.1

Microsoft Windows Server 2008, 2008 R2

Solaris 10

SUSE Linux Enterprise Server 11

SUSE Linux Enterprise Server 12 ppc64le

Ubuntu 16.04 (all architectures)

Removed agent platforms

Important: Before upgrading to this version, remove the `pe_repo::platform` class for the following 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 s390x

Enterprise Linux 7 s390x

Fedora 26, 27, 28, 29

Mac OS X 10.9, 10.12, 10.13

SUSE Linux Enterprise Server 11

SUSE Linux Enterprise Server 12 s390x

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.

Check out this video for more information about how Puppet works:

Quick start guide

The following pages are meant to teach you the basics of installing PE, adding nodes to your inventory, setting up your control repo, and running through an example task for managing webserver configurations (using Apache to manage a *nix machine or IIS to manage a Windows machine).

Note: This guide is intended as a simple overview to demonstrate basic PE setup and concepts. You'll likely use more features and need additional customizations described in detail elsewhere in the PE documentation.

- [Install PE](#) on page 66

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

- [Log in to the PE console](#) on page 72

The Puppet Enterprise (PE) console is a graphical interface where you can manage your infrastructure without relying on the command line.

- [Check the status of your primary server](#) on page 72

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

- [Add nodes to the inventory](#) on page 73

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 a Puppet agent installed, but can do things like run tasks and plans.

- [Add code and set up Code Manager](#) on page 74

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

- [Manage Apache configuration on *nix targets](#) on page 78

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy

relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

- [Manage IIS configuration on Windows targets](#) on page 84

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

- [Next steps](#) on page 91

Now that you have set up some basic automated configuration management with Puppet Enterprise (PE), here are some things you might want to do next.

Install PE

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

- [Install PE using installer tarball](#) on page 66

This installer employs default settings to install PE infrastructure components on a single node, creating a standard PE architecture. You can use a standard installation to try out PE with up to 10 nodes, or to manage up to 4,000 nodes. From there, you can scale up to the large or extra-large installation as your infrastructure grows, or customize your configuration as needed.

- [Install PE using PIM](#) on page 67

Puppet Installation Manager (PIM) supports the deployment of standard, large, and extra-large PE architectures. For an interactive experience, choose the guided installation and follow the step-by-step process in your terminal to configure and install the PE infrastructure you require. Alternatively, if you do not require guidance, you can create a JSON file containing your custom installation parameters, and run the installation from the command line.

Install PE using installer tarball

This installer employs default settings to install PE infrastructure components on a single node, creating a standard PE architecture. You can use a standard installation to try out PE with up to 10 nodes, or to manage up to 4,000 nodes. From there, you can scale up to the large or extra-large installation as your infrastructure grows, or customize your configuration as needed.

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

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

Important: The primary server can only run on a *nix machine. However, Windows machines can be Puppet agents, and you can manage them with your *nix primary server. Furthermore, you can operate your *nix primary server remotely from a Windows machine. To do this, before you install PE on your *nix primary server, you must configure an SSH client (such as [PuTTY](#)) with the hostname or IP address and port of the *nix machine that you'll use as your primary server. When you open an SSH session to install PE on the *nix primary server, log in as root or use sudo.

Related information

[What gets installed and where?](#) on page 116

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.

[Supported operating systems](#) on page 98

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

[Commands with elevated privileges](#) on page 30

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

[FIPS 140-2 enabled PE](#) on page 17

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.

[Verify the installation package](#) on page 125

This task is only required if your organization requires you to verify authenticity before installing packages. These steps explain how to use GnuPG (GPG) to verify the PE installation tarball.

Install PE from tarball

Before you begin

Review the [Hardware requirements for standard installations](#) on page 96 to make sure your system capacity can handle the standard PE installation.

Log in as root on your target primary server. If you're installing on a system that doesn't allow root login, you must use `sudo su -` to complete these steps.

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

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

2. To unpack the installation tarball, run:

```
tar -xzf <TARBALL_FILENAME>
```

3. From the installer directory, run `./puppet-enterprise-installer` and follow the CLI instructions to complete the installation.

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

After completing the standard installation, you can scale or customize your installation, if needed. For information and requirements for large and extra-large installations, go to [Supported architectures](#) on page 92 and [System requirements](#) on page 96. You can use [Configuration parameters](#) and the `pe.conf` file on page 129 to customize your installation.

Install PE using PIM

Puppet Installation Manager (PIM) supports the deployment of standard, large, and extra-large PE architectures. For an interactive experience, choose the guided installation and follow the step-by-step process in your terminal to configure and install the PE infrastructure you require. Alternatively, if you do not require guidance, you can create a JSON file containing your custom installation parameters, and run the installation from the command line.

Regardless of the installation process you choose, you can use PIM on a jump host to install PE infrastructure components on remote nodes that run a supported PE operating system. Alternatively, you can install PE locally by using PIM on a machine running a supported PE operating system. In this scenario, if you require additional infrastructure nodes to host PE components, your local machine can serve as a jump host.

PE infrastructure nodes are the hosts where PE components are installed. The following table lists the infrastructure nodes you can include in your installation when you use PIM to install PE:

PE infrastructure node	Description
Primary server (required)	Essential for a PE installation. Can host all components and services for smaller scale environments that include up to 2,000 nodes.

PE infrastructure node	Description
Primary server replica (optional)	To set up disaster recovery, install a replica of the primary server. If your primary server fails, the replica takes over to continue critical operations.
Database server	In an extra-large PE installation, a dedicated database server hosts a PostgreSQL instance containing the PuppetDB database.
Database server replica (optional)	Provides backup support during failovers.
Compilers	Compilers process Puppet code and convert it into catalogs that can be applied to agent nodes. The primary server can handle requests and compile catalogs for up to 2,000 agent nodes. In large and extra-large PE installations, dedicated compiler nodes help accelerate catalog compilation.

Related information

[Supported architectures](#) on page 92

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. When you install PE using the PE installer tarball, you begin with the standard configuration, and can then scale up by adding additional infrastructure components as needed. Alternatively, by using Puppet Installation Manager (beta) to install PE, you can start out with a standard, large, or extra-large configuration.

[What gets installed and where?](#) on page 116

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.

[Supported operating systems](#) on page 98

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

Install PE using the guided process

For an interactive experience, use the guided installation process. Based on information you provide about your environment and requirements, PIM automatically configures your PE installation.

Before you begin

- Ensure that you have the required access to the nodes where you want to install PE infrastructure.
 - To install the primary server locally on the machine where PIM is running, you must log in as the root user.
 - To install PE components on remote nodes, the machine running PIM must have SSH access to the target nodes, and the user executing the installation must have superuser privileges for those nodes.
- Ensure that Puppet is not already installed on any of the nodes where you want to install PE infrastructure.
- Check system requirements:
 - [Hardware requirements](#)
 - [Supported operating systems](#)
 - [Supported browsers](#)
 - [System configuration](#)

Important: Security-Enhanced Linux (SELinux) is enabled and enforced by default on Red Hat Enterprise Linux 9 (RHEL 9) operating systems. In order to use PIM, users must provide permission for PIM binary.

To install PE by using the PIM guided process:

1. Download PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Start the guided installation process.

In your terminal, navigate to the `pim` directory and run the following command:

```
./pim wizard
```

3. Follow the guided steps in your terminal to complete the installation.

If you require additional guidance during the installation process, you can view help content by pressing `Ctrl+H`.

Install PE with your defined parameters

If you know which PE infrastructure components you want to install and you do not require guidance, you can specify your installation parameters in a JSON file. Then use PIM to start the installation by running a single command.

Before you begin

- Ensure that you have the required access to the nodes where you want to install PE infrastructure.
 - To install the primary server locally on the machine where PIM is running, you must log in as the root user.
 - To install PE components on remote nodes, the machine running PIM must have SSH access to the target nodes, and the user executing the installation must have superuser privileges for those nodes. You can configure SSH, or use the `-b` flag to pass the SSH key or SSH credentials when you run the installation command.
- Ensure that Puppet is not already installed on any of the nodes where you want to install PE infrastructure.
- Check system requirements:
 - [Hardware requirements](#)
 - [Supported operating systems](#)
 - [Supported browsers](#)
 - [System configuration](#)

To install PE from the PIM command line:

1. Download PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Create a JSON file specifying the installation parameters you require.

For examples illustrating the JSON properties required for different PE architectures, see [Creating an installation parameters file](#).

3. Start the installation.

In your terminal, navigate to the `pim` directory and run one of the following commands, replacing `parameters.json` with the actual file name (including the file path, if necessary):

- To run the installation without debugging and without configuring SSH, use a command like the following example:

```
./pim install parameters.json
```

- To enable debug logging, add `-d` or `--debug`. For example:

```
./pim install parameters.json --debug
```

- To pass an SSH key or SSH credentials for accessing remote nodes, use the `-b` flag with the installation command as shown in the following examples:

```
./pim install -b user=root -b private-key=~/ssh/ssh_key params.json
```

```
./pim install -b user=root -b password=ssh_password params.json
```

4. Follow the CLI prompts to complete the installation process.

Note: PIM uses the Puppet Enterprise Administration Module (PEADM), which depends on Puppet Bolt, a tool for automating Puppet infrastructure maintenance tasks. When you run the `./pim install` command, PIM checks whether Bolt is present and, if necessary, provides the option to install Bolt.

Creating an installation parameters file

To install PE from the Puppet Installation Manager (PIM) command line, you must create a JSON file containing your installation parameters and pass that file with the `install` command. The JSON file defines your installation architecture, including the option for disaster recovery.

Important: Creating a JSON file containing installation parameters is not required if you use the guided installation process. With the guided process, PIM automatically configures your installation based on the information you provide about your environment and requirements.

Installation configuration examples

The following examples illustrate how to structure the JSON file for different PE configurations.

Installation parameters for an extra-large architecture with disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "replica_host": "pe-xl-core-2.lab1.puppet.vm",
  "replica_postgresql_host": "pe-xl-core-3.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}
```

Installation parameters for an extra-large architecture without disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
```

```

"compiler_hosts": [
  "pe-xl-compiler-0.lab1.puppet.vm",
  "pe-xl-compiler-1.lab1.puppet.vm"
],
"console_password": "puppetlabs",
"dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
"version": "2023.6.0"
}

```

Installation parameters for a large architecture with disaster recovery

```

{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
  "replica_host": "pe-l-core-2.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Installation parameters for a large architecture without disaster recovery

```

{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Installation parameters for a standard architecture with disaster recovery

```

{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "replica_host": "pe-core-2.lab1.puppet.vm",
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Installation parameters for a standard architecture without disaster recovery

```

{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Log in to the PE console

The Puppet Enterprise (PE) 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. In your browser, open the console by entering a URL referencing your primary server hostname. For example:

```
https://primary.example.com
```

Note: You'll receive a browser warning about an untrusted certificate because you were the signing authority for the console's certificate, and your PE deployment is not known to your browser as a valid signing authority. Ignore this warning and accept the certificate.

2. Log in with your admin username and password.

- If you used the PE installer tarball to install PE, log in to the console with the username `admin` and the password you created when installing. Keep track of this login because you'll need it later.
- If you used PIM to install PE, log in to the console with the admin username and password included in the JSON file containing the parameters used for your installation. Your installation parameters file has a name like `/success_install_<DATE_TIME_STAMP>.json`. When you are logged in successfully, change your admin password promptly for account security.

Next, check your primary server's status.

Related information

[Accessing the console](#) on page 265

The console is the web interface for Puppet Enterprise.

Check the status of your primary server

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

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

1. In the console, in the **Orchestration** section, click **Tasks**.
2. Click **Run a task** in the upper right corner of the **Tasks** page.
3. In the **Task** field, select `service` because you are checking the status of the primary server service.
4. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
5. Under **Task parameters**, enter 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 your primary server's hostname 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 the nodes that failed during the initial run.

Tip: You can filter run results by task name to find specific task runs.

View the task status and output on the **Jobs** page after the task is finished running.

Confirm that your primary server's status is *running* and *enabled*.

Next, use the console to [Add nodes to the inventory](#) on page 73.

To learn more about tasks, including how to install them from the Forge and how to write your own tasks, go to [Installing tasks](#) and [Writing tasks](#).

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 a Puppet agent installed, but can do things like run tasks and plans.

Add agent nodes

You can use the Puppet Enterprise (PE) console to install Puppet agents on Windows, *nix, and macOS nodes and add the nodes to your inventory. Agents help with configuration management by periodically checking in with the primary server for the desired configuration, detecting and correcting changes to resources the agent manages, and reporting information to the primary server about your infrastructure.

These steps demonstrate one method for [Installing agents](#) on page 145. For more detailed instructions, go to [Install agents from the console](#) on page 150.

1. In the PE console, go to **Nodes > Add nodes > Install agents**.
2. Select a transport method used to remotely install the agent on the target node.
 - **SSH** for *nix and macOS
 - **WinRM** for Windows
3. Enter the target host names and the credentials required to access them. You can specify multiple targets, but only one set of credentials.

Important: If you use an SSH key, include the begin and end tags.
4. Click **Add nodes** to install agents on the specified nodes. You can click **Installation job started** to view the task's job details.
Agents are installed on the target nodes and then they automatically submit certificate signing requests (CSRs) to the primary server.
5. Go to **Certificates > Unsigned certificates** and accept the CSRs.

Add agentless nodes

You can add nodes that don't or can't have a Puppet agent installed on them. Agentless automation allows you to do manage nodes that don't have software installed, such as updating a package or restarting a server on demand.

1. In the PE console, click **Nodes > Add nodes**.
2. Click **Connect over SSH or WinRM**.
3. Select a transport method.
 - **SSH** for *nix and macOS targets
 - **WinRM** for Windows targets
4. Enter target host names and the credentials required to access them. If you use an SSH key, include the begin and end tags.
5. Optional: Select additional [Transport configuration options](#) on page 434. For example, to customize the connection port number, select **Target Port** from the **Target options** drop-down list, enter the desired port number, and click **Add**.
6. Click **Add nodes**.

After adding agentless nodes to your PE inventory, they are added to PuppetDB, and you can view them on the **Nodes** page (in the console). Any nodes in your inventory can be added to the inventory node list when you set up a job to run tasks. To review a node's connection settings or remove an agentless node from the inventory, go to the **Connections** tab on the **Node details** page.

Add code and set up Code Manager

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

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

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

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

Related information

[How Code Manager works](#) on page 775

To automatically manage your environments and modules, Code Manager uses r10k and the file sync service to stage, commit, and sync your code.

Create a control repository from the Puppet template

To create a control repository (or control repo) that has the recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

The [Puppet control repo template](#) contains the necessary files to configure a functioning code management control repo plus helpful Puppet code examples, including:

- Basic code examples for setting up roles and profiles.
- A Puppetfile that references modules to manage content in your environments.
- An example Hiera configuration file and hieradata directory.
- A config_version script that tells you which version of code from your control repo was applied to your agents.
- An environment.conf file that implements the config_version script and a site-modules directory for roles, profiles, and custom modules.

In situations where you can't access the internet, or where organizational security policies prevent downloading modules from the Forge, you can [Create an empty control repo](#) on page 765 and add the necessary files to it.

To use the template, you must set up a private SSH key, copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

Important: The following steps assume you are using GitHub Enterprise with SSH. For more information and instructions for other version control hosts, such as GitLab or BitBucket, go to the [Puppet control-repo template README](#).

- To allow access to the control repo, generate a private SSH key without a password:

- To generate the key pair, run:

```
ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

- To allow the pe-puppet user to access the key, run:

```
puppet infrastructure configure
```

Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.

- Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as [GitHub](#), [BitBucket Server](#), or [GitLab](#)).

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

- In your Git user account or organization, create a repository named `control-repo`, and make sure a `README` is **not** automatically generated when you create the repo. Take note of the repo's SSH URL.

Important: Do not use an existing repo. The template requires a new, empty repo named `control-repo`.

- If you haven't already installed Git, run `yum install git`.
- To clone the Puppet `control-repo` template, run:

```
git clone https://github.com/puppetlabs/control-repo.git
```

- Change to the `control-repo` directory: `cd control-repo`
- Remove the template repo as the origin: `git remote remove origin`
- Set your control repo as the origin: `git remote add origin <SSH_URL_FOR_YOUR_CONTROL_REPO>`
- Push the contents of the `production` branch of the cloned control repo to your remote control repo: `git push origin production`

You now have a control repository based on the Puppet `control-repo` template. After configuring Code Manager, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, Code Manager detects and deploys your infrastructure changes.

By using the `control-repo` template, you now also have a Puppetfile to which you can add and manage content, like module code.

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

[Managing code with Code Manager](#) on page 774

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

[Add an environment](#) on page 767

Create new environments by creating branches based on your control repo's production branch.

Configure Code Manager

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

Enable Code Manager

Set parameters in the console to enable Code Manager and connect your primary server to your Git repository.

Before you begin

Set up an SSH key to permit the `pe-puppet` user to access your Git repositories. The SSH key must be:

- Owned by the `pe-puppet` user.
- Located on the primary server.
- Located in a directory the `pe-puppet` user has permission to view, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`.

These steps use the `puppet job` command. To use this command, you must have permission to run jobs and have access to the primary server.

1. In the console, click **Node groups**, locate the **PE Master** node group, and set these parameters for the `puppet_enterprise::profile::master` class:

- a) Set `code_manager_auto_configure` to `true` to enable Code Manager.
- b) For `r10k_remote`, enter a string that is a valid SSH URL for your Git control repository, such as `git@<YOUR.GIT.SERVER.COM>:puppet/control.git`.

Important: Some Git providers have additional requirements for enabling SSH access. For example, BitBucket requires `ssh://` at the beginning of the SSH URL (such as `ssh://git@<YOUR.GIT.SERVER.COM>:puppet/control.git`). See your provider's documentation for this information.

- c) For `r10k_private_key`, enter a string specifying the path to the SSH private key that permits the `pe-puppet` user to access your Git repositories, such as `" /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519 "`.

Important: If your PE installation includes disaster recovery, you must also set the `puppet_enterprise::profile::master::r10k_private_key` parameter in `pe.conf`. This ensures that the r10k private key is synced to your primary server replica.

2. Click **Commit**.

3. On the command line, run `puppet job run --nodes <NODE NAME>` where `<NODE NAME>` is the name of your primary server. For example:

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

Set up authentication for Code Manager.

Related information

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

[Run Puppet on demand from the CLI](#) on page 611

Use the `puppet job run` command to start an on-demand Puppet run to enforce changes on your agent nodes.

Set up authentication for Code Manager

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

Before requesting an authentication token, you must assign a user to the deployment role.

1. In the Puppet Enterprise (PE) console, create a deployment user.

Tip: Create a dedicated deployment user for Code Manager to use.

2. Add the deployment user to the **Code Deployers** role.

When you install PE, this role is automatically created with default permissions for code deployment and token lifetime management.

3. Click **Generate Password** to create a password for the deployment user.

Request an authentication token for deployments.

Related information

[Configure puppet-access](#) on page 303

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.

[Add a user to a user role](#) on page 279

When you add a user to a role, the user gains the permissions you assign to that role. A user can't do anything in PE until they have been assigned to at least one role. If users are assigned to multiple roles, they get all permissions from all roles they are assigned to.

[Assign a user group to a user role](#) on page 287

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

To securely deploy your code, request an authentication token for the deployment user.

The default lifetime for authentication tokens is one hour. You can use the `Override default expiry` permission set to change the token lifetime to a duration better suited for a long-running, automated process.

Use the `puppet-access` command to generate the authentication token.

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

Tip: You can specify additional settings in this command, such as the token file's location or your RBAC API URL, as explained in [Configuration file settings for puppet-access](#).

2. Enter the deployment user's username and password when prompted.

The generated token is stored in a file for later use. The default token storage location is `~/ .puppetlabs/token`. You can run `puppet-access show` to view the token.

Test the connection to the control repo.

Related information

[Set a token-specific lifetime](#) on page 308

If you want a token to have a different lifetime than the default lifetime, you can set a different lifetime when you generate the token. This allows you to keep one token for multiple sessions.

[Generate a token for use by a service](#) on page 307

If you need to generate a token that a Puppet Enterprise (PE) service can use, and the token doesn't need to be saved, use the `--print` option with the `puppet-access` command.

Test the connection and deploy your code

Make sure Code Manager can connect to your control repository, make a test deployment to a single environment, and then deploy code to all environments.

1. To test the connection to the control repo, run: `puppet-code deploy --dry-run`

If the control repo is set up properly, this command fetches and displays a list of environments in the control repo as well as the total number of environments.

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.

2. If the control repo connection works, test Code Manager by deploying a single environment. From the command line, run: `puppet-code deploy my_test_environment --wait`

The `--wait` flag returns results after the deployment is finished.

If Code Manager is configured correctly, this command deploys the test environment and returns deployment results with the SHA (a checksum for the content stored) for the control repo commit.

If the deployment does not work, review the Code Manager configuration steps or refer to [Troubleshooting](#) on page 855 for help.

3. After enabling and testing Code Manager, you can trigger Code Manager to deploy all environments. SSH into your primary server and run: `puppet-code deploy --all --wait`

You can also use `puppet-code deploy <ENVIRONMENT> --wait` to deploy a specific environment.

After making changes to your Puppetfile, such as adding a new module or creating a repo, you must deploy your code so Code Manager can recognize and start managing the content. You can trigger deployments from the command line, webhooks, or custom scripts.

Related information

[Triggering Code Manager on the command line](#) on page 795

Use the `puppet-code` command to trigger Code Manager from the command line and deploy your environments.

[Triggering Code Manager with a webhook](#) on page 801

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. Webhooks are the simplest way to trigger Code Manager.

[Triggering Code Manager with custom scripts](#) on page 803

Custom scripts are a good way to trigger deployments if you can't use webhooks. For example, if you have privately hosted Git repositories, custom notifications, or existing continuous integration systems (like Continuous Delivery for Puppet Enterprise (PE)).

Manage Apache configuration on *nix targets

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

Here we demonstrate how to distribute Apache services to a node group by installing the `apache` module, creating a node group for the Apache nodes, creating profiles to specify Apache and webserver configurations, bundling the profiles into a role, and assigning the role to the node group. Once this is done, PE distributes the role to the individual nodes and ensures the individual nodes have the Apache service and the desired configurations.

- [Install the apache module](#) on page 79

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice

working with modules, try installing the `puppetlabs/apache` module, which automates installing, configuring, and managing Apache services.

- [Set up Apache node groups](#) on page 80

Tell Puppet Enterprise (PE) your desired infrastructure configuration by creating *node groups* to categorize related nodes based on their function. Before you begin, decide which of your inventory nodes you want to have Apache services.

- [Organize webserver configurations with roles and profiles](#) on page 81

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

Install the apache module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the `puppetlabs/apache` module, which automates installing, configuring, and managing Apache services.

Before you begin

[Install PE](#) on page 66 and at least one [*nix agent node](#) before installing the `apache` module.

Tip: You can write your own modules or download pre-built modules from the [Forge](#). While you can use any module on the Forge, PE customers can take advantage of supported modules. These modules are designed to facilitate common services, and they are tested and maintained by Puppet.

1. Go to the [Apache module](#) page on the Forge.
2. Select **r10k or Code Manager** as the **Installation Method**, and follow the instructions to add the module declaration to your Puppetfile.

By default, Code Manager installs the latest version and disables automatic updates; however, you can specify options to install a different version or keep the module current with the latest version. To automatically update the module when a new version is released, specify `:latest` (such as `mod 'puppetlabs/apache', :latest`). To install a specific version of the module and prevent automatic updates, specify the version number as a string (such as `mod 'puppetlabs/apache', '5.4.0'`).

3. Make sure your Puppetfile includes module declarations for the `puppetlabs/stdlib` and `puppetlabs(concat` modules, which are dependencies of the `apache` module. Dependencies for each module are listed on the **Dependencies** tab on the module's Forge page, and you can specify the desired version in the same way you did for the primary module.

For example, this code installs version 5.4.0 of the `apache` module, installs the module's dependencies, and prevents automatic updates (due to specified version numbers):

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

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

You installed the `apache` module. Installing a module makes it available in PE so you can use it to manage nodes.

After installing a module, you can run tasks included in the module from the PE console. The `apache` module contains a task that allows you to perform Apache service functions. To view or run the `apache` task in the PE console, go to the **Run** section, under **Task**, and enter `apache` in the **Task** field.

If the `apache` task doesn't appear, try refreshing the console in the browser. If it still does not appear, check that the `puppetlabs/apache` module is in your Puppetfile and try again.

To continue managing Apache configuration on *nix targets, [Set up Apache node groups](#) on page 80.

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

Set up Apache node groups

Tell Puppet Enterprise (PE) your desired infrastructure configuration by creating *node groups* to categorize related nodes based on their function. Before you begin, decide which of your inventory nodes you want to have Apache services.

Important: Setting up a node group does not install software or configure the individual nodes in the node group. A node group is a categorical designation for related nodes (in this case, nodes running the Apache module), and you use the node group to manage nodes in bulk. After creating a node group, you must apply a role to the group so the nodes have some specifications to inherit.

Create your classification group

A classification node group is a parent group for other node groups that contain classification data. You only need to set up the classification node group once.

The classification group distinguishes classification node groups from environment node groups.

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

You created the classification group.

Create a node group and add it as a child of the classification group.

Create your apache node group

After creating the parent classification group, make the apache node group as a child of the classification group.

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

You have created a node group to categorize the nodes you want to have Apache services.

Identify nodes from your inventory that you want to run Apache on and add them to the apache node group.

Add nodes to the apache node group

Once you determine which nodes from your inventory belong in your apache node group, pin the nodes to the group. Pinning adds nodes to a group one at a time. If you have a lot of nodes, you can create rules in the console to dynamically add nodes to a node group.

[Best practices for classifying node groups](#) on page 441 can help you decide which nodes belong in your apache node group.

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

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

You have added your apache nodes to the apache node group.

Run Puppet.

Related information

[Making changes to node groups](#) on page 449

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

[Grouping and classifying nodes](#) on page 440

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

Run Puppet on your Apache nodes

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

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

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

After running Puppet on a node group, the nodes in the group automatically check in with PE every 30 minutes to confirm that their configuration matches what you've specified for this group. PE corrects the configuration if it doesn't match.

You must create roles and profiles to define specific applications, services, and configurations you want the nodes in this group to receive, as explained in [Organize webserver configurations with roles and profiles](#) on page 81.

Important: Until you assign a role to a node group, the group has no defined configuration for the nodes to inherit.

Organize webserver configurations with roles and profiles

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

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

To illustrate roles and profiles, these steps demonstrate how to:

- Define a profile that configures virtual webhost (vhost) to serve the example.com website with a firewall rule.
- Create a role to contain the profile.
- Assign the role to the apache node group.

This creates a base structure where, if you had additional websites to serve, you would create additional profiles for those sites. When you have multiple profiles, you can combine profiles within roles or create unique roles for each profile.

Because this example adds a firewall rule, make sure you add the [puppetlabs/firewall](#) module to your Puppetfile, following the same process you used to [Install the apache module](#) on page 79. Remember to add the firewall modules dependencies (`puppetlabs/stdlib`), such as:

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

Related information

[The roles and profiles method](#) on page 486

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

Set up your prerequisites

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

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

Write a profile for your Apache vhost

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

Before you begin

Make sure you have:

- Installed the `puppetlabs/apache` module, the `puppetlabs/firewall` module, and their dependencies from the Forge.
- Created the `role` and `profile` modules, as explained in [Set up your prerequisites](#).

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

1. In the `profile` module, create the following directories and `.pp` file:

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

- Paste this Puppet code into the `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 you vhost's main page has content. Finally, there is a firewall rule that only allows traffic from the ports set in the `$ports` array.

You can add your own code to the profile as needed. For more information, go to these Forge pages:

- [Apache module README](#)
- [Apache module Reference](#)
- [Firewall module README](#)
- [Firewall module Reference](#)

Set data for the profile

Hiera is a configuration method that allows you to set defaults in your code or override defaults (in certain circumstances). Use it to refine profile data.

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

With Hiera structured data, you can set up a four-layer hierarchy consisting of:

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

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

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

```
# cat /etc/puppetlabs/code/environments/production/data/stage/dev.yaml
```

```
---
profile::webserver::example::content: "Hello from dev\n"
profile::webserver::example::ports:
  - '8080'
```

You'd use this code to make 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 a brief introduction to what you can do with structured data in Hiera. To learn more about setting up hierarchical data, see [Getting started with Hiera](#).

Write a role for your Apache webserver

Roles contain sets of profiles. To write roles, think about the machines you're managing and decide what else they need in addition to the webserver profile.

This example shows how to write a role by combining profiles. In this example, assume you want all nodes in your apache node group to use the webserver profile you just wrote, and that your organization assigns all machines (including workstations) a profile called `profile::base` that manages basic policies and uses some conditional logic to include operating-system-specific configuration.

1. In your control repo, open the `.pp` file for the `role` module. If it doesn't exist, create the necessary directories and file, such as:

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

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

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

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

Assign the role to nodes

Assign the `exampleserver` role to the node group containing the nodes that you want to have the Apache vhost configuration you wrote in the `webserver::example` profile.

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

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

Now, the `apache` node group manages your Apache vhost based on the rules you wrote in your `webserver` profile. When the nodes check in with PE, PE distributes the role (and the contained profiles) to the individual nodes and ensures the individual nodes have the Apache service and the desired configurations.

Manage IIS configuration on Windows targets

Your nodes need applications and services to perform their intended functionality. Not all nodes need all software, and different types of nodes require different software configurations. Puppet Enterprise (PE) help you deploy

relevant software and configurations to your nodes by grouping related nodes and allowing you to specify relevant software and configurations for each node group.

Here we demonstrate how to distribute IIS services to a node group by installing the `iis` module, creating a node group for the IIS nodes, creating profiles to specify IIS and webserver configurations, bundling the profiles into a role, and assigning the role to the node group. Once this is done, PE distributes the role to the individual nodes and ensures the individual nodes have the IIS service and the desired configurations.

- [Install the iis module](#) on page 85

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the `puppetlabs/iis` module, which automates installing, configuring, and managing IIS services.

- [Set up IIS node groups](#) on page 86

Tell Puppet Enterprise (PE) your desired infrastructure configuration by grouping and classifying nodes based on their function. Before you begin, decide which of your inventory nodes you want to have IIS services.

- [Organize webserver configurations with roles and profiles](#) on page 87

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

Install the `iis` module

Modules are self-contained, shareable bundles of code and data. Each module manages a specific task in your infrastructure, such as installing and configuring a piece of software. With Puppet Enterprise (PE), a lot of your infrastructure is supported by modules, so it is important to learn how to install, build, and use them. To practice working with modules, try installing the `puppetlabs/iis` module, which automates installing, configuring, and managing IIS services.

Before you begin

[Install PE](#) on page 66 and at least one [Windows agent node](#) before installing the `iis` module.

Tip: You can write your own modules or download pre-built modules from the [Forge](#). While you can use any module on the Forge, PE customers can take advantage of supported modules. These modules are designed to facilitate common services, and they are tested and maintained by Puppet.

1. Go to the [IIS module](#) page on the Forge.
2. Select **r10k or Code Manager** as the **Installation Method**, and follow the instructions to add the module declaration to your Puppetfile.

By default, Code Manager installs the latest version and disables automatic updates; however, you can specify options to install a different version or keep the module current with the latest version. To automatically update the module when a new version is released, specify `:latest` (such as `mod 'puppetlabs/iis', :latest`). To install a specific version of the module and prevent automatic updates, specify the version number as a string (such as `mod 'puppetlabs/iis', '7.0.0'`).

3. Make sure your Puppetfile includes module declarations for the `puppetlabs/pwshlib` module, which is a dependency of the `iis` module. Dependencies for each module are listed on the **Dependencies** tab on the module's Forge page, and you can specify the desired version in the same way you did for the primary module.

For example, this code installs version 7.0.0 of the `iis` module, installs the module's dependency, and prevents automatic updates (due to specified version numbers):

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

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

You installed the `iis` module. Installing a module makes it available in PE so you can use it to manage nodes.

To continue managing IIS configuration on Windows targets, [Set up IIS node groups](#) on page 86.

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

Set up IIS node groups

Tell Puppet Enterprise (PE) your desired infrastructure configuration by grouping and classifying nodes based on their function. Before you begin, decide which of your inventory nodes you want to have IIS services.

Important: Setting up a node group does not install software or configure the individual nodes in the node group. A node group is a categorical designation for related nodes (in this case, nodes running the IIS module), and you use the node group to manage nodes in bulk. After creating a node group, you must apply a role to the group so the nodes have some specifications to inherit.

Create your classification group

A classification node group is a parent group for other node groups that contain classification data. You only need to set up the classification node group once.

The classification group distinguishes classification node groups from environment node groups.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the new node group:

- **Parent name:** All Nodes
- **Group name:** All Classification
- **Environment:** production
- **Environment group:** Do not select

3. Click **Add**.

You created the classification group.

Create a node group and add it as a child of the classification group.

Create your iis node group

After creating the parent classification group, make the **iis** node group as a child of the classification group.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the **iis** node group:

- **Parent name:** All Classification
- **Group name:** iis
- **Environment:** production
- **Environment group:** Do not select

3. Click **Add**.

You have created a node group to categorize the nodes you want to have IIS services. Next, determine which nodes from your inventory you want to run IIS on and add them to the **iis** node group.

Identify nodes from your inventory that you want to run IIS on and add them to the **iis** node group.

Add nodes to the **iis** node group

Once you determine which nodes from your inventory belong in your **iis** node group, pin the nodes to the group. Pinning adds nodes to a group one at a time. If you have a lot of nodes, you can create rules in the console to dynamically add nodes to a node group.

[Best practices for classifying node groups](#) on page 441 can help you decide which nodes belong in your **iis** node group.

1. In the console, click **Node groups** and select the **iis** node group.

2. On the **Rules** tab, under **Certname**, enter the certname of a node you want to add to the **iis** node group.

3. Click **Pin node**.
4. Repeat to add more nodes to the `iis` group.
5. When you are done pinning nodes, commit changes.

You have added your `iis` nodes to the `iis` node group.

Run Puppet.

Related information

[Making changes to node groups](#) on page 449

You can edit or remove node groups, remove nodes or classes from node groups, and edit or remove parameters and variables.

[Grouping and classifying nodes](#) on page 440

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

Run Puppet on your IIS nodes

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

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Under **Run options**, do not select anything.
3. From the list of target types, select **Node group**.
4. In the **Choose 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 it on all nodes or only the nodes that failed during the initial run.

After running Puppet on a node group, the nodes in the group automatically check in with PE every 30 minutes to confirm that their configuration matches what you've specified for this group. PE corrects the configuration if it doesn't match.

You must create roles and profiles to define specific applications, services, and configurations you want the nodes in this group to receive, as explained in [Organize webserver configurations with roles and profiles](#) on page 87.

Important: Until you assign a role to a node group, the group has no defined configuration for the nodes to inherit.

Organize webserver configurations with roles and profiles

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

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

To illustrate roles and profiles, these steps demonstrate how to:

- Define a profile that configures the `example.com` website and includes a firewall rule.

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

- Create a role to contain the profile.
- Assign the role to the `iis` node group.

This creates a base structure where, if you had additional websites to serve, you would create additional profiles for those sites. When you have multiple profiles, you can combine profiles within roles or create unique roles for each profile.

Because this example adds a firewall rule, make sure you add the [puppet/windows_firewall](#) module to your Puppetfile, following the same process you used to [Install the iis module](#) on page 85. Remember to add the firewall modules dependencies (`puppetlabs/stdlib` and `puppetlabs/registry`), such as:

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

Related information

[The roles and profiles method](#) on page 486

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

Set up your prerequisites

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

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

Write a profile for your IIS website

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

Before you begin

Make sure you have:

- Installed the `puppetlabs/iis` module, the `puppet/windows_firewall` module, and their dependencies from the Forge.
- Created the `role` and `profile` modules, as explained in [Set up your prerequisites](#).

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

1. In the `profile` module, create the following directories and `.pp` file:

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

- Paste this Puppet code into the `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 the site's main page has content. Finally, there is a firewall rule that only allows traffic from the ports set in the `$port` setting.

You can add your own code to the profile as needed. For more information, go to these Forge pages:

- [IIS module README](#) © 2024 Puppet, Inc., a Perforce company
- [IIS module Reference](#)

Set data for the profile

Hiera is a configuration method that allows you to set defaults in your code or override defaults (in certain circumstances). Use it to refine profile data.

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

With Hiera structured data, you can set up a four-layer hierarchy consisting of:

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

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

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

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

You'd use this code to make webservers in the production environment listen to all interfaces:

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

This is a brief introduction to what you can do with structured data in Hiera. To learn more about setting up hierarchical data, see [Getting started with Hiera](#).

Write a role for your IIS website

Roles contain sets of profiles. To write roles, think about the machines you're managing and decide what else they need in addition to the `webserver` profile.

This example shows how to write a role by combining profiles. In this example, assume you want all nodes in your `iis` node group to use the `webserver` profile you just wrote, and that your organization assigns all machines (including workstations) a profile called `profile::base` that manages basic policies and uses some conditional logic to include operating-system-specific configuration.

1. In your control repo, open the `.pp` file for the `role` module. If it doesn't exist, create the necessary directories and file, such as:

```
site-modules\role\manifests\exampleserver.pp
```

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

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

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

Assign the role to nodes

Assign the `exampleserver` role to the node group containing the nodes that you want to have the `iis_site` configuration you wrote in the `webserver::example` profile.

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

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

Now, the `iis` node group manages your `iis_site` website based on the rules you wrote in your `webserver` profile. When the nodes check in with PE, PE distributes the role (and the contained profiles) to the individual nodes and ensures the individual nodes have the IIS service and the desired configurations.

Next steps

Now that you have set up some basic automated configuration management with Puppet Enterprise (PE), here are some things you might want 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 \(PDK\)](#).
- [Configure Puppet Enterprise](#) to fine-tune console performance, orchestration services, Java, proxy settings, and other aspects of your PE installation.
- Learn more about [Tasks in PE](#) on page 614 and [Plans in PE](#) on page 645.
- Refer to [Managing access](#) on page 268 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](#).
- Check out our [YouTube channel](#) to learn more about Puppet.
- Explore other parts of the [PE documentation](#).

Installing

A typical Puppet Enterprise (PE) deployment 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 92

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. When you install PE using the PE installer tarball, you begin with the standard configuration, and can then scale up by adding additional infrastructure components as needed. Alternatively, by using Puppet Installation Manager (beta) to install PE, you can start out with a standard, large, or extra-large configuration.

- [System requirements](#) on page 96

Refer to these system requirements for Puppet Enterprise installations.

- [What gets installed and where?](#) on page 116

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 PE](#) on page 124

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

- [Purchasing and installing a license key](#) on page 144

A Puppet Enterprise (PE) license includes access to Security Compliance Management (formerly Comply) and Continuous Delivery, both of which can be installed and used after you have installed PE. To unlock additional premium features including Security Compliance Enforcement (formerly CEM) and advanced Impact Analysis capabilities in Continuous Delivery, you can [contact our sales team](#).

- [Installing agents](#) on page 145

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

- [Installing compilers](#) on page 165

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 client tools](#) on page 172

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 176

Puppet Enterprise (PE) includes a script for uninstalling. You can uninstall infrastructure nodes or uninstall the agent 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. When you install PE using the PE installer tarball, you begin with the standard configuration, and can then scale up by adding additional infrastructure components as needed. Alternatively, by using Puppet Installation Manager (beta) to install PE, you can start out with a standard, large, or extra-large configuration.

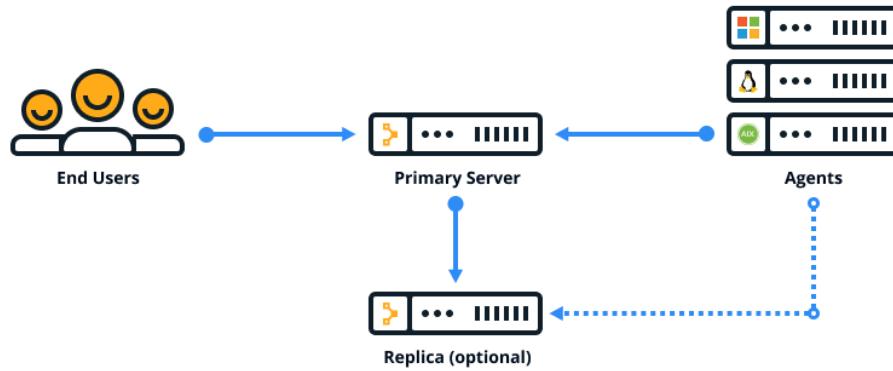
Existing customer deployments in the field might use other configurations, but for the best performance, scalability, and support, we recommend using one of our three defined architectures unless specifically advised otherwise by Puppet Support personnel. For legacy architectures, we document only upgrade procedures – not installation instructions – in order to support existing customers.

Tip: For guidance about deploying PE in global, multi-region, or multi-network segment scenarios, see the [Multi-region Reference Architectures](#) article.

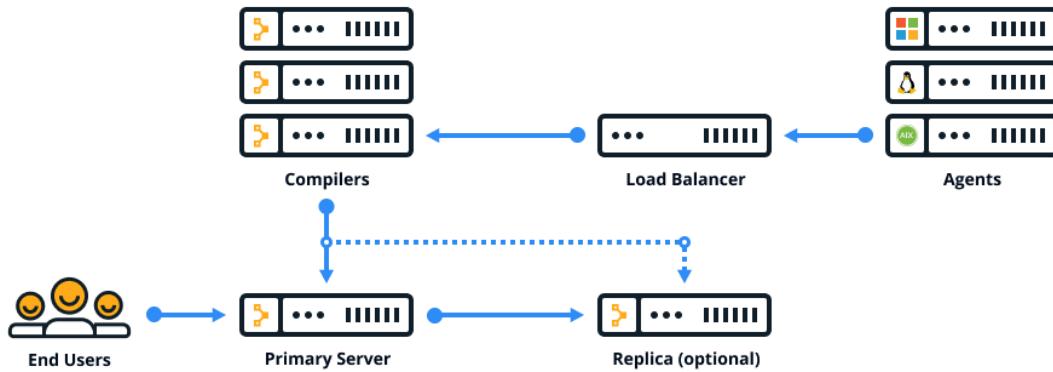
Configuration	Description	Node limit
Standard installation (Recommended)	All infrastructure components are installed on the primary server. This installation type is the easiest to install, upgrade, and troubleshoot.	Up to 2,000
Large installation	Similar to a standard installation, plus one or more compilers and a load balancer which help distribute the agent catalog compilation workload.	2,000–20,000

Configuration	Description	Node limit
Extra-large installation	Similar to a large installation, plus a database server hosting a PE-PostgreSQL instance for the PuppetDB database.	20,000+

Standard installation

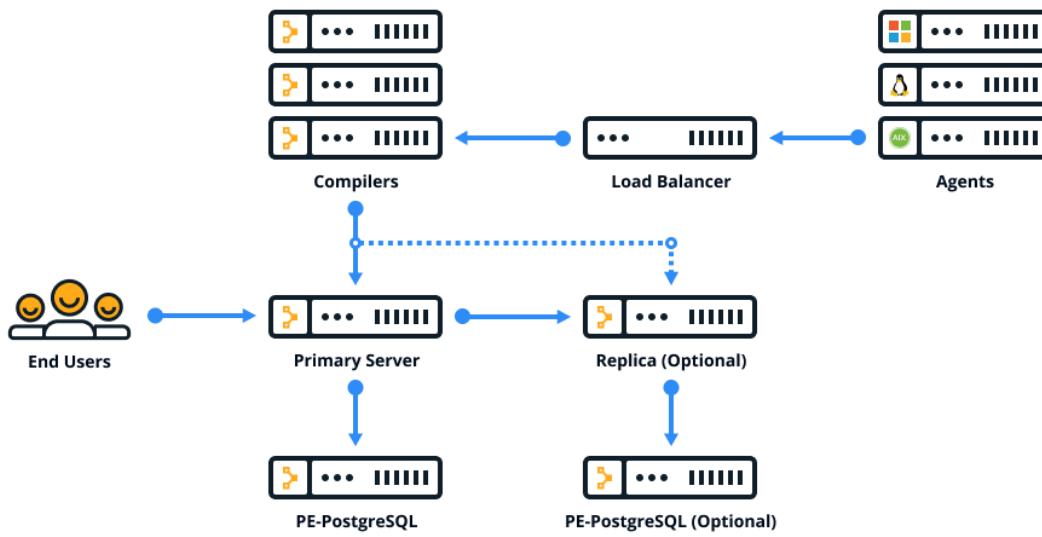


Large installation



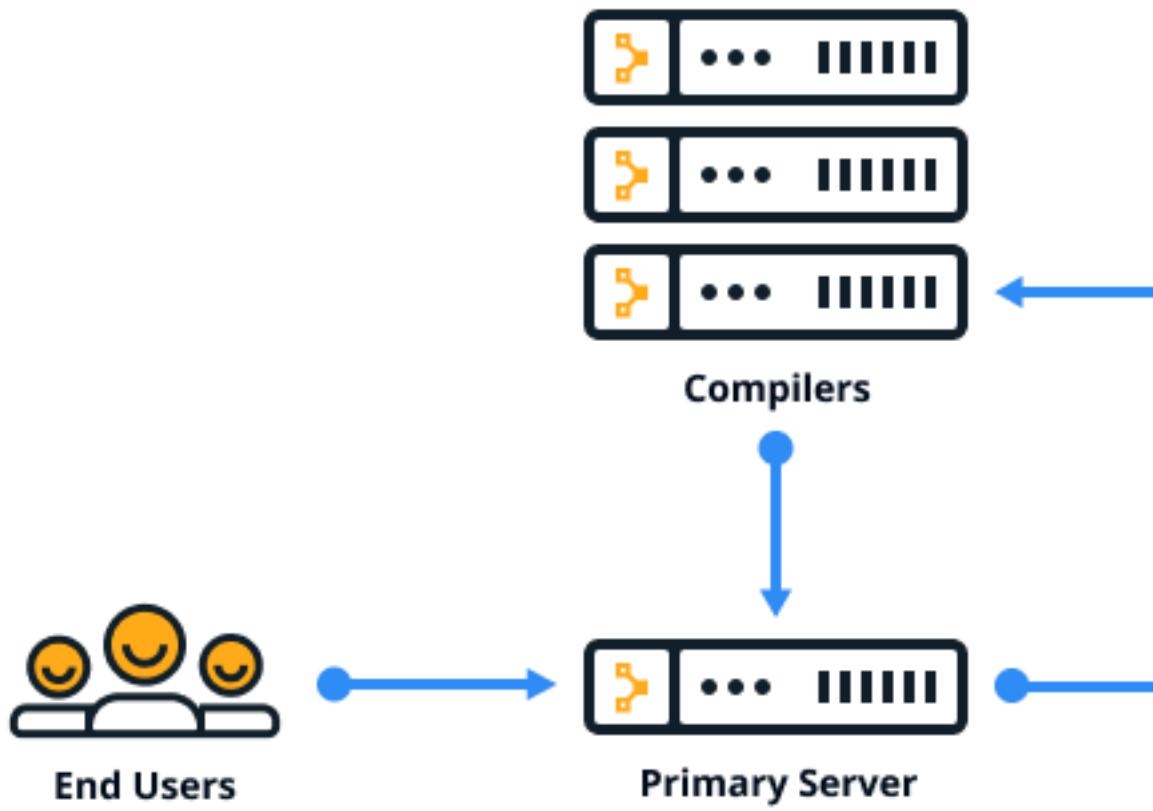
Extra-large installation

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 or Puppet Professional Services.



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 96

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 98

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

- [Supported browsers](#) on page 104

The following browsers are supported for use with the console.

- [System configuration](#) on page 104

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

[Tune infrastructure nodes](#) on page 203

Use these guidelines to configure your Puppet Enterprise (PE) installation to maximize use of available system resources (CPU and RAM).

[Puppet Enterprise metrics and status monitoring](#) on page 399

You can use Puppet Enterprise (PE) metrics and status monitoring for your own performance tuning or provide the information to Support for troubleshooting.

[View and manage Puppet Server metrics](#) on page 401

Puppet Server tracks performance and status metrics you can use to monitor server health and performance over time.

Hardware requirements for standard installations

These are the minimum hardware requirements for the primary server in a standard architecture with up to 2,500 nodes.

Node volume	Cores	RAM	/opt/	/var/
Trial use	2	8 GB	20 GB	24 GB
11–100	6	10 GB	50 GB	24 GB
101–500	8	12 GB	50 GB	24 GB
501–1,000	10	16 GB	50 GB	24 GB
1,000–2,500	12	24 GB	50 GB	24 GB

- **Trial mode:** Although the m5.large instance type is sufficient for trial use, it is not supported. A minimum of four cores is required for production workloads.
- **/opt/ storage requirements:** The database should not exceed 50% of /opt/ to allow for future upgrades.

- /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 your technical account manager or Puppet Professional Services to talk about optimizing your setup for your specific requirements.

Hardware requirements for cloud deployments

Ensure that your primary server meets the minimum hardware requirements for cloud deployments. Cloud deployments use a standard architecture with up to 2,500 nodes.

Node volume	Amazon Web Services (AWS)	Azure
Trial use	m5.large	D2 v4
11–100	c5.2xlarge	F8s v2
101–500	c5.2xlarge	F8s v2
501–1,000	c5.2xlarge	F8s v2
1,000–2,500	c5.4xlarge	F16s v2

Azure requirements are not currently tested by Puppet, but are presented here as best guidance based on comparable AWS testing.

Supported operating systems

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

Supported operating systems and devices

You can install PE and the agent on these supported platforms.

For details about platform support lifecycles and planned end-of-life support, see [Platform support lifecycle](#) on the Puppet website.

Important: If you implement Linux hardening techniques, consider customizing your settings, including but not limited to the following:

- **SELinux:** Grant exceptions for Puppet and the PXP agent to allow these services to run effectively.
- **File Access Policy Daemon (fapolicyd):** Grant exceptions for PE services to prevent potential restrictions.
- **umask:** Ensure your operating system's default umask is set to 022 or less restrictive. A more restrictive setting can lead to unintended failures, as Puppet users might be denied access to necessary files.

Primary server platforms

The PE primary server can be installed on the following operating systems. All primary server platforms require an x86_64 architecture (or amd64 for Ubuntu).



CAUTION: Major primary server OS upgrades (such as Ubuntu 20.04 to 22.04) require [Back up and restore PE](#) on page 846.

Operating system	Versions
AlmaLinux	8, 9
Amazon Linux	2
Oracle Linux	7, 8
Red Hat Enterprise Linux	7, 8, 9
Red Hat Enterprise Linux (FIPS 140-2 compliant)	7, 8
Rocky Linux	8, 9
SUSE Linux Enterprise Server	7, 15
Ubuntu (General Availability kernels)	18.04, 20.04, 22.04

Agent platforms

The agent can be installed on these operating systems and architectures.



CAUTION: Major agent OS upgrades (such as Ubuntu 20.04 to 22.04) require reinstalling the puppet-agent package (as explained in [Installing agents](#) on page 145) and reinstalling any Ruby plugins/gems that were added at /opt/puppetlabs/puppet/bin/gem.

Operating system	Versions	Architecture
AIX	7.1, 7.2, 7.3	POWER
	<p>Note: We support only technology levels that are still under support from IBM.</p>	
AlmaLinux	8, 9	<ul style="list-style-type: none"> 8: x86_64, aarch64, ppc64le 9: x86_64, aarch64
Amazon Linux	2, 2023	<ul style="list-style-type: none"> 2: aarch64ARM64 2023: aarch64, amd64
CentOS	6, 7	<ul style="list-style-type: none"> 6: x86_64, i386 7: x86_64
Debian	Buster (10), Bullseye (11), Bookworm (12)	<ul style="list-style-type: none"> 10: amd64 11: amd64, aarch64 12: amd64, aarch64
Fedora	36, 40	<ul style="list-style-type: none"> 36: x86_64 40: x86_64
macOS	11, 12, 13, 14	<ul style="list-style-type: none"> 11: x86_64 12: x86_64, M1 13: x86_64, ARM 14: x86_64, ARM
Microsoft Windows	10, 11	<ul style="list-style-type: none"> 10: x86, x64 11: x64 <p>For FIPS 140-2 compliant Microsoft Windows, use version 10 with x64 architecture.</p>
Microsoft Windows Server	2012, 2012 R2, 2012 R2 Core, 2016, 2016 Core, 2019, 2019 Core, 2022, 2016 FIPS	x64 <p>For FIPS 140-2 compliant Microsoft Windows Server, use 2012 R2 or 2012 R2 Core.</p>

Operating system	Versions	Architecture
Oracle Linux	6, 7, 8, 9	<ul style="list-style-type: none"> • 6: x86_64, i386 • 7: x86_64 • 8: x86_64, aarch64, ppc64le • 9: x86_64
Red Hat Enterprise Linux	6, 7, 8, 9	<ul style="list-style-type: none"> • 6: x86_64, i386 • 7: x86_64 • 8: x86_64, aarch64, ppc64le • 9: x86_64, ARM64, ppc64le <p>For FIPS 140-2 compliant RHEL, use version 7, 8 or 9 with x86_64 architecture.</p>
Rocky Linux	8, 9	<ul style="list-style-type: none"> • 8: x86_64, aarch64, ppc64le • 9: x86_64, aarch64
Scientific Linux	6, 7	<ul style="list-style-type: none"> • 6: x86_64, i386 • 7: x86_64
Solaris	10, 11	<ul style="list-style-type: none"> • 10: SPARC, i386 • 11: SPARC, x86_64
SUSE Linux Enterprise Server	12, 15	x86_64
Ubuntu (General Availability kernels)	18.04, 20.04, 22.04, 24.04	<ul style="list-style-type: none"> • 18.04: amd64, aarch64 • 20.04: amd64, aarch64 • 22.04: amd64, aarch64 • 24.04: amd64, aarch64

Platform dependencies

When you install PE or an agent, certain package dependencies are required to ensure the node is operational.

In most cases, dependencies are automatically set up during installation. You might need to manually install dependencies in these cases:

- If you're installing on AIX or Solaris.
- If the node doesn't have internet access.

Note: Some operating systems require an active subscription with the vendor's package management system (for example, the Red Hat Network) to install dependencies.

CentOS dependencies

	All nodes	Primary server
cronie	x	x
dmidecode	x	x
libxml2	x	x
logrotate	x	x
net-tools	x	x
pciutils	x	x
tar	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

	All nodes	Primary server
cronie	x	x
dmidecode	x	x
libxml2	x	x
logrotate	x	x
net-tools	x	x
pciutils	x	x
tar	x	x
which	x	x
zlib	x	x
curl		x
libjpeg		x
libtool-ltdl (versions 7 and later)		x
libxslt		x
mailcap		x
initscripts		x

SUSE Linux Enterprise Server dependencies

Tip: If you encounter problems installing dependencies, inspect the error messages for packages that require other SUSE Linux Enterprise Server packaging modules to be enabled, and use `zypper package-search <PACKAGE NAME>` to locate them for manual installation.

	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
tar	x	x
zlib	x	x
curl		x
db43		x
libjpeg		x
unixODBC		x

Ubuntu 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
tar	x	x
zlib	x	x
ca-certificates-java		x
curl		x
file		x
libcap2		x
libgtk2.0-0		x
libjpeg62		x

	All nodes	Primary server
libmagic1		x
libossp-uuid16		x
libpcre3		x
libxslt1.1		x
mime-support		x
perl		x

AIX dependencies

AIX is a supported platform for the agent only. Before installing the agent on AIX systems, install these packages.

- bash
- curl
- openssl
- readline
- tar
- zlib

For information about installing these packages, see [AIX Toolbox for Open Source Software](#).

Restriction: For OpenSSL, you must use the version provided by IBM Marketing Registration Services (MRS). For more information, see the IBM support docs about [Downloading and Installing or Upgrading OpenSSL and OpenSSH](#).

Solaris dependencies and limitations

Solaris support is agent only.

For Solaris 11 these packages are required:

- system/readline
- system/library/gcc-45-runtime
- library/security/openssl
- tar

These packages are available in the Solaris release repository, which is enabled by default in version 11. The installer automatically installs these packages; however, if the release repository is not enabled, the packages must be installed manually.

Upgrade your operating system with PE installed

If you have PE installed, take extra precautions before performing a major upgrade of your machine's operating system.

Performing major upgrades of your operating system with PE installed can cause errors and issues with PE. A major operating system upgrade is an upgrade to a new whole version, such as an upgrade from RHEL 6.0 to 7.0; it does not refer to a minor version upgrade, like RHEL 6.5 to 6.6. Major upgrades typically require a new version of PE.

1. Back up your databases and other PE files.
2. Perform a complete uninstall (using the `-p` and `-d` uninstaller options).
3. Upgrade your operating system.
4. Install PE.
5. Restore your backup.

Related information

[Back up your infrastructure](#) on page 847

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

[Restore your infrastructure](#) on page 848

Use the restore process when you migrate your primary server to a new operating system or to a new host. You can also use the restore process to recover your installation after a system failure.

[Uninstalling](#) on page 176

Puppet Enterprise (PE) includes a script for uninstalling. You can uninstall infrastructure nodes or uninstall the agent from agent nodes.

[Installing PE](#) on page 124

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

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	Current version as of release

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.

URL	Enables
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 (infrastructure nodes) to accommodate the software so it doesn't slow your performance.
- Some antivirus and antimalware software defaults to using port 8081, which is the same port PuppetDB uses. When installing the software, consider which port it uses so it doesn't conflict with PuppetDB communications.
- For agents, you can exclude `C:\ProgramData\PuppetLabs\pe_patch` if your antivirus is holding a lock on log files and causing patching failures.

Related information

[Tune infrastructure nodes](#) on page 203

Use these guidelines to configure your Puppet Enterprise (PE) installation to maximize use of available system resources (CPU and RAM).

[The PuppetDB default port conflicts with another service](#) on page 863

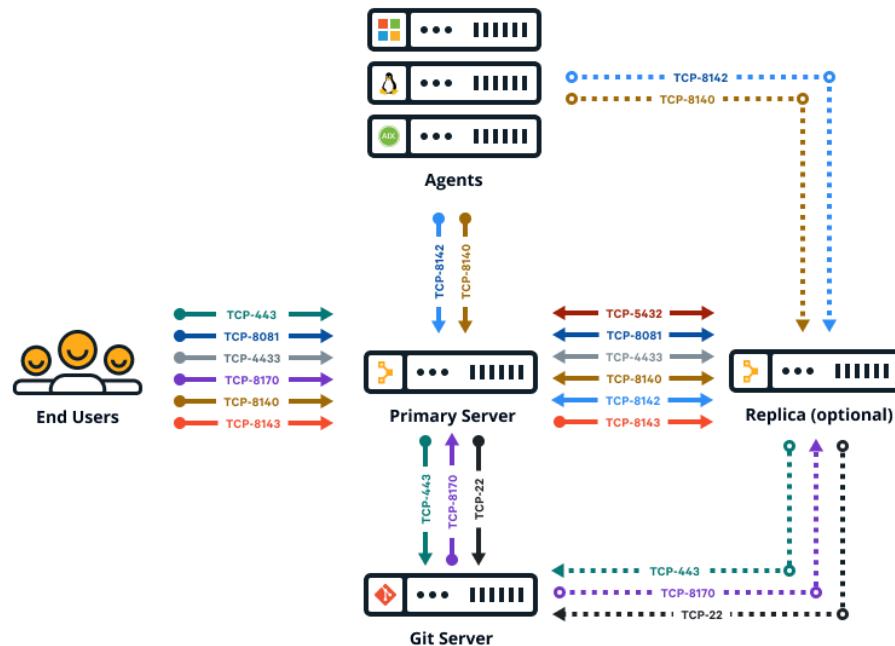
By default, PuppetDB communicates over port 8081. In some cases, this might conflict with other services, such as McAfee ePolicy Orchestrator.

Firewall configuration

Follow these guidelines for firewall configuration based on your installation type.

Firewall configuration for standard installations

The port requirements for standard installations are described.



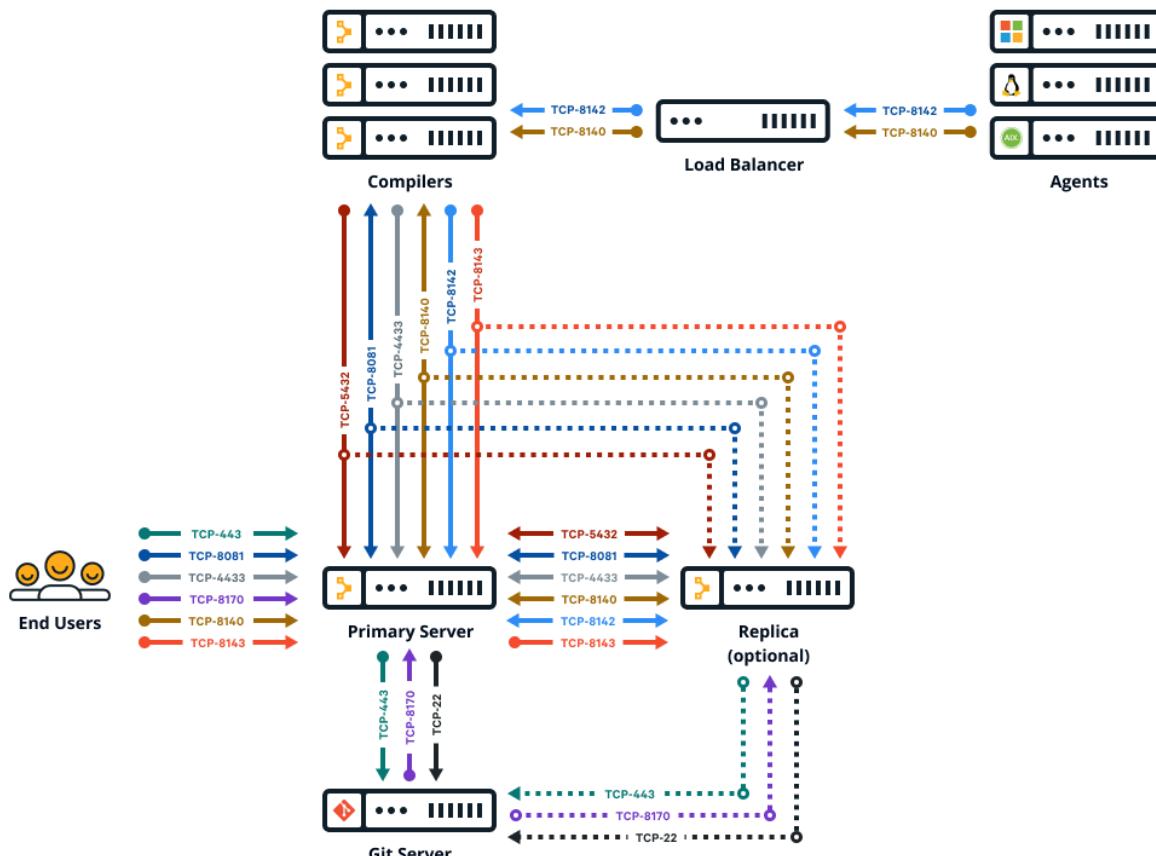
Port	Use	Virtual private networks for cloud deployments
22	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via SSH. 	External

Port	Use	Virtual private networks for cloud deployments
443	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via HTTPS. This port provides host access to the console. The console accepts HTTPS traffic from end users on this port. Classifier group: PE Console 	External
4433	<ul style="list-style-type: none"> This port is used as a classifier and console services API endpoint. The primary server communicates with the console over this port. Classifier group: PE Console 	External
5432	<ul style="list-style-type: none"> This port is used to replicate PostgreSQL data between the primary server and the replica. 	Internal
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic and requests on this port. The primary server and console send traffic to PuppetDB on this port. PuppetDB status checks are sent over this port. Classifier group: PE PuppetDB 	Internal

Port	Use	Virtual private networks for cloud deployments
8140	<ul style="list-style-type: none"> The primary server uses this port to accept inbound traffic and 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 	Internal
8142	<ul style="list-style-type: none"> Orchestrator and the Run Puppet button use this port on the primary server to accept inbound traffic and responses from agents via the Puppet Execution Protocol agent. Classifier group: PE Orchestrator 	Internal
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 	Internal
8170	<ul style="list-style-type: none"> Code Manager uses this port to deploy environments, run webhooks, and make API calls. 	Internal

Firewall configuration for large installations

These are the port requirements for large installations with compilers.



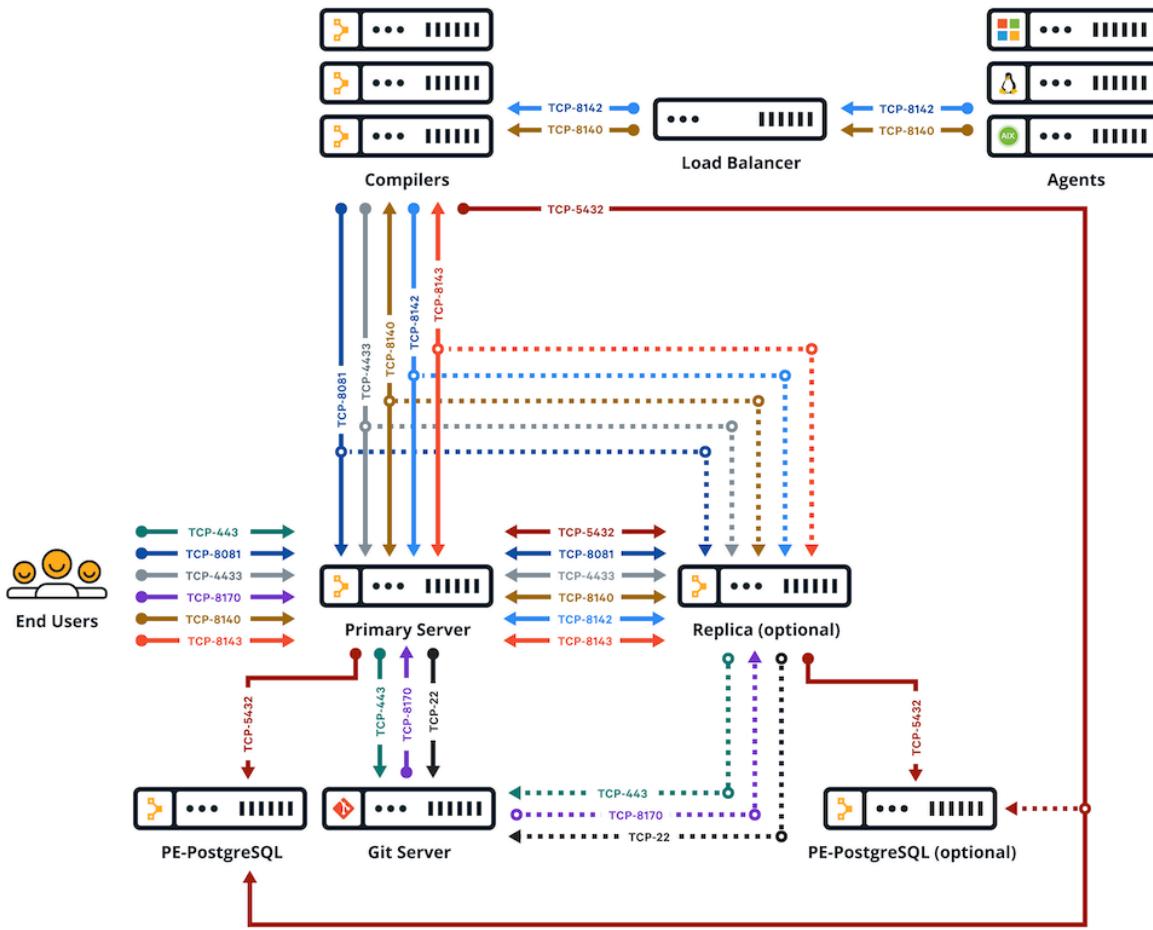
Port	Use
22	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via SSH.
443	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via HTTPS. This port provides host access to the console. The console accepts HTTPS traffic from end users on this port. Classifier group: PE Console

Port	Use
4433	<ul style="list-style-type: none"> This port is used as a classifier and 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 the replica. The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic and 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 and 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 and 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	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.



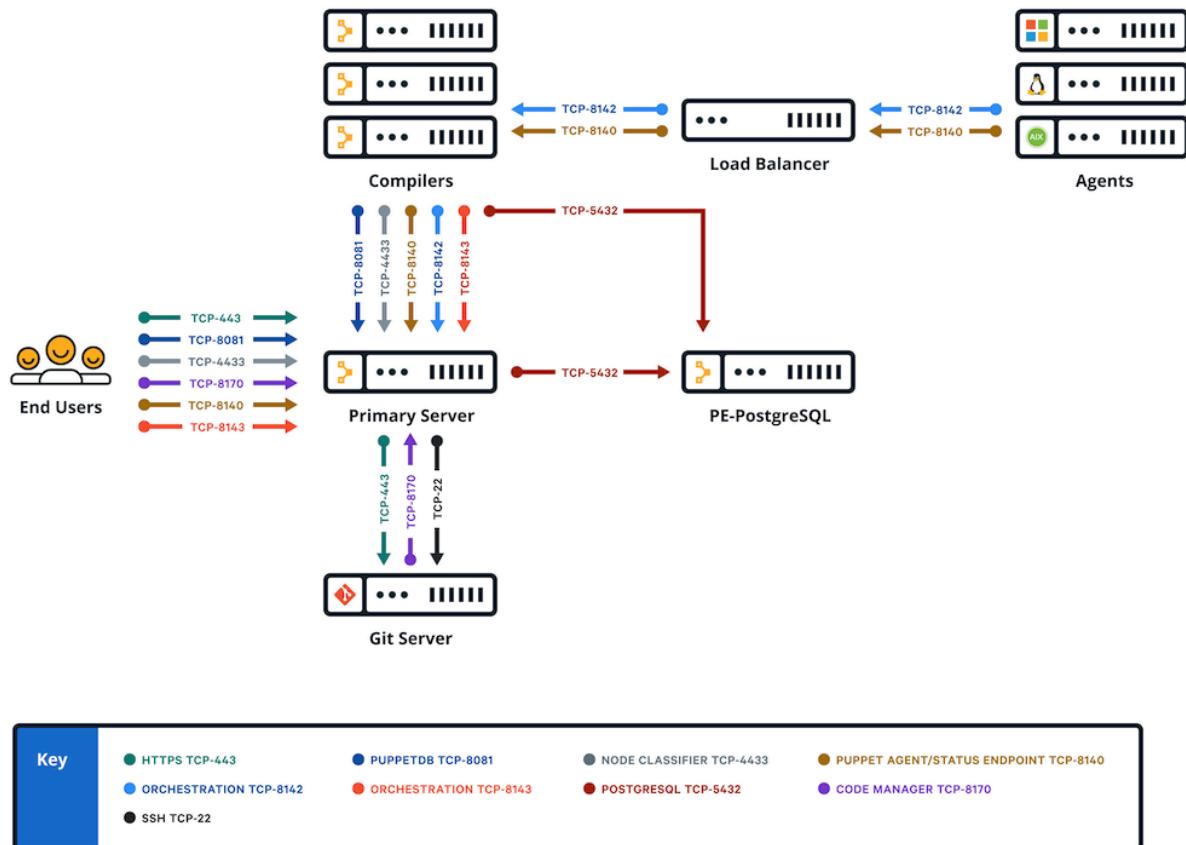
Port	Use
22	<ul style="list-style-type: none">Code Manager uses this port to tell a git to clone and fetch content via SSH.
443	<ul style="list-style-type: none">Code Manager uses this port to tell a git to clone and fetch content via HTTPS.This port provides host access to the console.The console accepts HTTPS traffic from end users on this port.Classifier group: PE Console

Port	Use
4433	<ul style="list-style-type: none"> This port is used as a classifier and 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 primary server and replica use this port to replicate PostgreSQL data on PE-PostgreSQL nodes. The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic and 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 and 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 and 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
22	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via SSH.

Port	Use
443	<ul style="list-style-type: none"> Code Manager uses this port to tell a git to clone and fetch content via HTTPS. This port provides host access to the console. The console accepts HTTPS traffic from end users on this port. Classifier group: PE Console
4433	<ul style="list-style-type: none"> This port is used as a classifier and 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. The PuppetDB service running on compilers uses this port to communicate with PE-PostgreSQL.
8081	<ul style="list-style-type: none"> PuppetDB accepts traffic and 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 and 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 and 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 2021.y and 2019.8.z releases.

The functional components of the software are separated between those packaged with the agent and those packaged on the server side (which also includes the agent).

Note: PE also installs other dependencies, as documented in the system requirements.

The following table shows the components that are installed on all agent nodes.

Tip: 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	Factor	Ruby	OpenSSL
2021.7.10	7.35.0	4.8.0	<ul style="list-style-type: none"> MRI Ruby: 2.7.8 (Puppet agent) JRuby: 9.3.14.0 (Puppet server) 	1.1.1v
2021.7.9	7.32.1	4.8.0	<ul style="list-style-type: none"> MRI Ruby: 2.7.8 (Puppet agent) JRuby: 9.3.14.0 (Puppet server) 	1.1.1v
2021.7.8	7.30.0	4.7.0	2.7.8	1.1.1v
2021.7.7	7.28.0	4.5.2	2.7.8	1.1.1v

PE Version	Puppet and the Puppet agent	Facter	Ruby	OpenSSL
2021.7.6	7.27.0	4.5.1	2.7.8	1.1.1v
2021.7.5	7.26.0	4.4.3	2.7.8	1.1.1v
2021.7.4	7.24.0	4.3.1	2.7.7	1.1.1t
2021.7.3	7.24.0	4.3.1	2.7.7	1.1.1t
2021.7.2	7.21.0	4.2.14	2.7.7	1.1.1q
2021.7.1	7.20.0	4.2.13	2.7.6	1.1.1q
2021.7.0	7.18.0	4.2.11	2.7.6	1.1.1q
2021.6	7.16.0	4.2.8	2.7.6	1.1.1n
2021.5	7.14.0	4.2.7	2.7.5	1.1.1l
2021.4	7.12.1	4.2.5	2.7.3	1.1.1l
2021.3	7.9.0	4.2.2	2.7.3	1.1.1k
2021.2	7.8.0	4.2.1	2.7.3	1.1.1k
2021.1	7.6.1	4.1.1	2.7.3	1.1.1i
2021.0	7.4.1	4.0.51	2.7.2	1.1.1i
2019.8.12	6.28.0	3.14.24	2.5.9	1.1.1q
2019.8.11	6.27.0	3.14.23	2.5.9	1.1.1n
2019.8.10	6.26.0	3.14.22	2.5.9	1.1.1l
2019.8.9	6.25.1	3.14.21	2.5.9	1.1.1l
2019.8.8	6.24.0	3.14.19	2.5.9	1.1.1k
2019.8.7	6.23.0	3.14.18	2.5.9	1.1.1k
2019.8.6	6.22.1	3.14.17	2.5.9	1.1.1g
2019.8.5	6.21.1	3.14.16	2.5.8	1.1.1i
2019.8.4	6.19.1	3.14.14	2.5.8	1.1.1g
2019.8.3	6.19.1	3.14.14	2.5.8	1.1.1g
2019.8.1	6.17.0	3.14.12	2.5.8	1.1.1g
2019.8	6.16.0	3.14.11	2.5.8	1.1.1g

The following table shows components that are installed on server nodes.

PE Version	Puppet Server	PuppetDB	r10k	Bolt Services	Agentless Catalog Executor (ACE) Services	PostgreSQL	Java	Nginx
2021.7.10	7.17.4	7.21.0	3.16.2	3.30.0	1.2.4	14.13	11.0.26.4.26.2	
2021.7.9	7.17.2	7.19.1	3.16.2	3.30.0	1.2.4	14.13	11.0.24.8.26.2	
2021.7.8	7.17.1	7.18.0	3.16.1	3.29.0	1.2.4	14.11	11.0.23.9.25.1	
2021.7.7	7.15.0	7.16.0	3.16.0	3.27.4	1.2.4	14.10	11.0.22.7.25.1	

PE Version	Puppet Server	PuppetDB	r10k	Bolt Services	Agentless Catalog Executor (ACE) Services	PostgreSQL	Java	Nginx
2021.7.6	7.14.0	7.15.0	3.16.0	3.27.4	1.2.4	14.8	11.0.21.9.25.1	
2021.7.5	7.13.1	7.14.0	3.16.0	3.27.2	1.2.4	14.8	11.0.20.8.25.1	
2021.7.4	7.11.0	7.13.0	3.15.4	3.27.1	1.2.4	14.5	11.0.19.6.22.0	
2021.7.3	7.11.0	7.13.0	3.15.4	3.27.1	1.2.4	14.5	11.0.19.6.22.0	
2021.7.2	7.9.4	7.12.1	3.15.4	3.26.2	1.2.4	14.5	11.0.17.8.22.0	
2021.7.1	7.9.2	7.11.2	3.15.2	3.26.1	1.2.4	14.5	11.0.6 1.22.0	
2021.7.0	7.9.0	7.11.1	3.15.1	3.26.1	1.2.4	14.5	11.0 1.22.0	
2021.6	7.7.0	7.10.1	3.14.0	3.22.1	1.2.4	14.1	11.0 1.21.0	
2021.5	7.6.0	7.9.2	3.14.0	3.21.0	1.2.4	11.13	11.0 1.21.0	
2021.4	7.4.2	7.7.1	3.13.0	3.20.0	1.2.4	11.13	11.0 1.21.0	
2021.3	7.2.1	7.5.2	3.10.0	3.13.0	1.2.4	11.13	11.0 1.21.0	
2021.2	7.2.0	7.4.1	3.9.2	3.10.0	1.2.4	11.11	11.0 1.21.0	
2021.1	7.1.2	7.3.1	3.9.0	3.7.1	1.2.4	11.11	11.0 1.19.6	
2021.0	7.0.3	7.1.0	3.8.0	3.0.0	1.2.2	11.10	11.0 1.19.6	
2019.8.12	6.20.0	6.22.1	3.15.1	3.25.0	1.2.4	11.17	11.0 1.22.0	
2019.8.11	6.19.0	6.21.0	3.14.0	3.22.1	1.2.4	11.15	11.0 1.21.0	
2019.8.10	6.18.0	6.20.2	3.14.0	3.21.0	1.2.4	11.13	11.0 1.21.0	
2019.8.9	6.17.1	6.19.1	3.13.0	3.20.0	1.2.4	11.13	11.0 1.21.0	
2019.8.8	6.16.1	6.18.2	3.10.0	3.13.0	1.2.4	11.13	11.0 1.17.0	
2019.8.7	6.16.0	6.17.0	3.9.2	3.10.0	1.2.4	11.11	11.0 1.21.0	
2019.8.6	6.15.3	6.16.1	3.9.0	3.7.1	1.2.4	11.11	11.0 1.19.6	
2019.8.5	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 Hiera 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 task`.
 - **Puppet Access** — Users can generate tokens to authenticate their access to certain command line tools and API endpoints.
 - **Code Manager CLI** — The `puppet-code` command allows you to trigger Code Manager from the command line to deploy your environments.
 - **PuppetDB CLI** — This a tool for working with PuppetDB, including building queries and handling exports.
- **Module tool** — The module tool is used to access and create modules, which are reusable chunks of Puppet code users have written to automate configuration and deployment tasks. For more information, and to access modules, visit the Forge.

- **Console** — The console is the web user interface for PE. The console provides tools to view and edit resources on your nodes, view reports and activity graphs, and more.

Databases installed

PE installs several default databases, all of which use PostgreSQL as a database backend.

The PE PostgreSQL database includes the following databases:

Database	Contents
pe-activity	Activity data from the classifier, including who, what, and when
pe-classifier	Classification data, all node group information
pe-inventory	Connection information and credentials for agentless node connections
pe-orchestrator	Orchestrator data, including details about job runs
pe-puppetdb	PuppetDB data, including exported resources, catalogs, facts, and reports
pe-rbac	RBAC data, including users, permissions, and AD/LDAP info

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.
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 Agentless Catalog Executor (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.

User	Definition
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

Code Manager access log

Location: /var/log/puppetlabs/puppetserver/code-manager-access.log

File sync access log

Location: /var/log/puppetlabs/puppetserver/file-sync-access.log

Puppet Communications Protocol (PCP) broker log

This is the log file for PCP brokers on compilers.

Location: /var/log/puppetlabs/puppetserver/pcp-broker.log

General Puppet Server log

This is where the primary server logs its activity, including compilation errors and deprecation warnings.

Location: /var/log/puppetlabs/puppetserver/puppetserver.log

Puppet Server access log

Location: /var/log/puppetlabs/puppetserver/puppetserver-access.log

Puppet Server daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/puppetserver/puppetserver-daemon.log

Puppet Server status log

Location: /var/log/puppetlabs/puppetserver/puppetserver-status.log

Agent logs

The agent log locations depend on the agent node's operating system.

On *nix nodes, the agent service logs activity to the syslog service. The node's operating system and syslog configuration determines where these messages are saved. The default locations are as follows:

- Linux: /var/log/messages
- macOS: /var/log/system.log
- Solaris: /var/adm/messages

On Windows nodes, the agent service logs its activity to the Event Log. Browse the Event Viewer to view those messages. You might need to enable [Logging and debugging](#) on page 870.

Console and console services logs

General console services log

Location: /var/log/puppetlabs/console-services/console-services.log

Console services API access log

Location: /var/log/puppetlabs/console-services/console-services-api-access.log

Console services access log

Location: /var/log/puppetlabs/console-services-access.log

Console services daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/console-services-daemon.log

NGINX access log

Location: /var/log/puppetlabs/nginx/access.log

NGINX error log

Contains console errors that aren't logged elsewhere and errors related to NGINX.

Location: /var/log/puppetlabs/nginx/error.log

Installer logs

HTTP log

Contains web requests sent to the installer.

Only exists on machines from which a web-based installation was performed.

Location: /var/log/puppetlabs/installer/http.log

Orchestrator info log

Contains run details about `puppet infrastructure` commands that use the orchestrator. This includes commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.

Location: /var/log/puppetlabs/installer/orchestrator_info.log

Last installer run logs, by hostname

Contains the contents of the last installer run.

There can be multiple log files, labeled by hostname.

Location: /var/log/puppetlabs/installer/install_log.lastrun.<HOSTNAME>.log

Installer operation logs, by timestamp

Captures operations performed during installation and any errors that occurred.

There can be multiple log files, labeled by timestamp.

/var/log/puppetlabs/installer/installer-<TIMESTAMP>.log

Disaster recovery command logs, by action, timestamp, and description

Contains details about disaster recovery command execution.

There can be multiple log files for each command because each action triggers multiple Puppet runs (Some on the primary server and some on the replica).

Location: /var/log/puppetlabs/installer/<ACTION-NAME>_<TIMESTAMP>_<RUN-DESCRIPTION>.log

Bolt info log

Can be valuable when [Troubleshooting disaster recovery](#) on page 859.

Location: /var/log/puppetlabs/installer/bolt_info.log

Database logs

Database logs include PostgreSQL and PuppetDB logs.

PostgreSQL startup log

Can be valuable when [Troubleshooting the databases](#) on page 862.

Location: /var/log/puppetlabs/postgresql/14/pgstartup.log

PostgreSQL daily logs, by weekday

There is one log file for each day of the week. Log file names use short names, such as Mon for Monday, Tue for Tuesday, and so on.

Location: /var/log/puppetlabs/postgresql/14/postgresql-<WEEKDAY>.log

General PuppetDB log

Location: /var/log/puppetlabs/puppetdb/puppetdb.log

PuppetDB access log

Location: /var/log/puppetlabs/puppetdb/puppetdb-access.log

PuppetDB status log

Location: /var/log/puppetlabs/puppetdb/puppetdb-status.log

Orchestration logs

Orchestrator logs include orchestration services and related components, such as PXP agent and Bolt server.

Aggregate node count log

Location: /var/log/puppetlabs/orchestration-services/aggregate-node-count.log

Puppet Communications Protocol (PCP) broker log

This is the log file for PCP brokers on the primary server.

Location: /var/log/puppetlabs/orchestration-services/pcp-broker.log

Puppet Communications Protocol (PCP) broker access log

Location: /var/log/puppetlabs/orchestration-services/pcp-broker-access.log

Orchestration services access log

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-access.log

Orchestration services daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-daemon.log

Orchestration services status log

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-status.log

Puppet Execution Protocol (PXP) agent log

*nix location: /var/log/puppetlabs/pxp-agent/pxp-agent.log

Windows location: C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log

Bolt server log

Can be valuable when [Troubleshooting connections between components](#) on page 860.

Location: /var/log/puppetlabs/bolt-server/bolt-server.log

Node inventory service log

Location: `/var/log/puppetlabs/orchestration-services/orchestration-services.log`

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`. You can learn more about the certificate authority service on the [PE software architecture](#) on page 7 page.

Related information

[SSL and certificates](#) on page 836

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.

[Use an independent intermediate certificate authority](#) on page 841

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 843

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.

Secret key file installed

During installation, the software generates secret key files that are used to encrypt and decrypt sensitive data.

The inventory service secret key is used to encrypt and decrypt sensitive data stored in the inventory service. This key is stored at:

```
/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json
```

The console services secret key is used to encrypt and decrypt passwords used for LDAP connections. This key is stored at:

```
/etc/puppetlabs/console-services/conf.d/secrets/keys.json
```

Installing PE

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

- [Install PE using the installer tarball](#) on page 125

This installer employs default settings to install PE infrastructure components on a single node, creating a standard PE architecture. You can use a standard installation to try out PE with up to 10 nodes, or to manage up to 4,000 nodes. From there, you can scale up to the large or extra-large installation as your infrastructure grows, or customize your configuration as needed.

- [Install PE using PIM](#) on page 140

Puppet Installation Manager (PIM) supports the deployment of standard, large, and extra-large PE architectures.

For an interactive experience, choose the guided installation and follow the step-by-step process in your terminal to configure and install the PE infrastructure you require. Alternatively, if you do not require guidance, you can create a JSON file containing your custom installation parameters, and run the installation from the command line.

Install PE using the installer tarball

This installer employs default settings to install PE infrastructure components on a single node, creating a standard PE architecture. You can use a standard installation to try out PE with up to 10 nodes, or to manage up to 4,000 nodes. From there, you can scale up to the large or extra-large installation as your infrastructure grows, or customize your configuration as needed.

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

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

Important: The primary server can only run on a *nix machine. However, Windows machines can be Puppet agents, and you can manage them with your *nix primary server. Furthermore, you can operate your *nix primary server remotely from a Windows machine. To do this, before you install PE on your *nix primary server, you must configure an SSH client (such as PuTTY) with the hostname or IP address and port of the *nix machine that you'll use as your primary server. When you open an SSH session to install PE on the *nix primary server, log in as root or use sudo.

To install a FIPS-enabled PE primary server, install the appropriate FIPS-enabled PE tarball (such as `puppet-enterprise-2021.7.10-redhatfips-7-x86_64.tar`) on a third-party [Supported operating system](#) with FIPS mode enabled. The node must be configured with sufficient available entropy for the installation process to succeed.

Related information

[What gets installed and where?](#) on page 116

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.

[Supported operating systems](#) on page 98

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

[Commands with elevated privileges](#) on page 30

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

[FIPS 140-2 enabled PE](#) on page 17

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.

Verify the installation package

This task is only required if your organization requires you to verify authenticity before installing packages. These steps explain how to use GnuPG (GPG) to verify the PE installation tarball.

Before you begin

You must have GnuPG (GPG) installed to be able to sign for the release key. GPG is an open source program you can use to safely encrypt and sign digital communications. You can download GPG from the [GnuPG website](#) or use your package management system to install it by running something like: `yum install gnupg`

1. Download the tarball appropriate to your operating system and architecture.

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

2. To import the Puppet public key, run:

```
uri='https://downloads.puppet.com/puppet-gpg-signing-key-20250406.pub'
curl "$uri" | gpg --import
```

Tip: For general information about forming curl commands, go to [Using example commands](#) on page 28.

3. To print the key fingerprint, run:

```
gpg --fingerprint 0x4528B6CD9E61EF26
```

This command returns the primary key fingerprint. For example:

```
D681 1ED3 ADEE B844 1AF5 AA8F 4528 B6CD 9E61 EF26
```

4. Download the **GPG SIGNATURE .asc** file corresponding to your PE tarball. You can find links to these files on the [PE Download page](#).
5. To verify the installation package release signature, run:

```
gpg --verify puppet-enterprise-<VERSION>-<PLATFORM>.tar.gz.asc
```

The `gpg --verify` command returns something similar to:

```
gpg: Signature made <DATE_AND_TIME>
gpg: using RSA key <KEY_ID>
gpg: Good signature from "Puppet, Inc. Release Key (Puppet, Inc. Release
Key) <release@puppet.com>"
```

Tip:

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

If you receive a warning that the key is not certified with a trusted signature, this means you haven't told GPG to trust the imported key. Refer to the [GPG documentation](#) for more information.

If you received the `Good signature` message, you can proceed to unpack the installation tarball and complete the installation, as outlined in [Install PE from tarball](#) on page 126.

Install PE from tarball

Before you begin

Review the [Hardware requirements for standard installations](#) on page 96 to make sure your system capacity can handle the standard PE installation.

Log in as root on your target primary server. If you're installing on a system that doesn't allow root login, you must use `sudo su -` to complete these steps.

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.

- To unpack the installation tarball, run:

```
tar -xzf <TARBALL_FILENAME>
```

- From the installer directory, run `./puppet-enterprise-installer` and follow the CLI instructions to complete the installation.
- Optional: Restart the shell to use client tool commands.

After completing the standard installation, you can scale or customize your installation, if needed. For information and requirements for large and extra-large installations, go to [Supported architectures](#) on page 92 and [System requirements](#) on page 96. You can use [Configuration parameters and the pe.conf file](#) on page 129 to customize your installation.

Launch an AWS image

You can launch a cloud PE image from the Amazon Web Services (AWS) console or an AWS software development kit (SDK) or by using third-party tools.

- Launch a PE image and specify the details for your deployment:

- EC2 instance type** – The image size used for your deployment. For recommendations, see [Hardware requirements for standard installations](#) on page 96.
- EC2 VPC and subnet** – The VPC or subnet in which to deploy your image.
- EC2 security group** – The security group policy to use for your deployment.

To control access to the image, a key pair is created.

- Connect to the image by using your new key pair and the username `puppetadmin`:

```
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

SSH keys are automatically provisioned, and no password is required.

- Wait for the image to start and for PE configuration to be completed.

To track progress, run the `check_status.sh` script:

```
sudo /opt/puppetlabs/cloud/bin/check_status.sh --wait
```

- Specify a console admin password:

```
sudo /opt/puppetlabs/puppet/bin/puppet infrastructure <console_password>
```

Console access is disabled until the password is set.

- Using a web browser, connect to the console at `https://<PRIMARY_HOSTNAME>`, accept the console's certificate, and log in with the username `admin` and the password that you specified during installation.

Tip: The console uses an SSL certificate created by your local Puppet certificate authority. Because this authority is specific to your site, web browsers don't know it or trust it, and you must add a security exception to access the console.

The console indicates that your primary server is actively managed by displaying the following message:

```
1 Nodes run in enforcement.
```

Your primary server is now ready to manage nodes.

Launch an Azure image

You can launch a cloud PE image from the Microsoft Azure Portal, PowerShell, or a software development kit (SDK) from [Ruby](#), [Python](#), [Go](#), or [Java](#).

1. Launch a PE image and specify the details for your deployment:

- **Resource Group** – Creates a resource group or reuses an existing group.
- **Location** – The location for the resource group. If you use an existing resource group, the location must match the resource group's location.
- **Admin Password** – The password for the admin user. If you select Secure Shell (SSH) authentication, the password you specify is used as a backup authentication method.
- **VM Size** – The size used for your deployment. For recommendations, see [Hardware requirements for standard installations](#) on page 96.
- **Admin User Name** – The username for logging in with SSH.
- **Authentication Type** – `sshPublicKey` or `password`.
- **SSH Public Key** – The SSH public key, if you select SSH as your authentication type.
- **Virtual Network New Or Existing** – Creates a virtual network or uses an existing network. If you select `existing`, you don't have to enter an address prefix or subnet.
- **Public IP Address New Or Existing** – new to specify a static IP address. If the IP is dynamic and the virtual machine (VM) is restarted, you won't be able to access the console, because the console uses the initial public IP address.
- **Public IP Address Domain Name Label** – The prefix of the fully qualified domain name used by the VM.
- **Storage Account New Or Existing** – Creates a storage account or uses an existing account. If you select `existing`, you don't have to enter an account type.

To control access to the image, a new key pair is created.

2. Connect to the image by using your new key pair and the username `puppetadmin`:

```
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

SSH keys are automatically provisioned, and no password is required.

3. Wait for the image to start and for PE configuration to be completed.

To track progress, run the `check_status.sh` script:

```
sudo /opt/puppetlabs/cloud/bin/check_status.sh --wait
```

4. Specify a console admin password:

```
sudo /opt/puppetlabs/puppet/bin/puppet infrastructure <console_password>
```

Console access is disabled until the password is set.

5. Using a web browser, connect to the console at `https://<PRIMARY_HOSTNAME>`, accept the console's certificate, and log in with the username `admin` and the password that you specified during installation.

Tip: The console uses an SSL certificate created by your local Puppet certificate authority. Because this authority is specific to your site, web browsers don't know it or trust it, and you must add a security exception to access the console.

The console indicates that your primary server is actively managed by displaying the following message:

```
1 Nodes run in enforcement.
```

Your primary server is now ready to manage nodes.

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 Puppet Enterprise (PE). A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

Tip: You can use a custom `pe.conf` file when installing PE by running: `./puppet-enterprise-installer -c <PATH_TO_pe.conf>`

The following table lists the value types you can use in the `pe.conf` file, along with examples of each type:

Type	Parameter-value format example
FQDN	<code>"puppet_enterprise::puppet_master_host": "primary.example.com"</code>
String	<code>"console_admin_password": "mypassword"</code>
Array	<code>["puppet", "puppetlb-01.example.com"]</code>
Boolean	<code>"puppet_enterprise::profile::orchestrator::run_se": true</code>
	Restriction: The only valid Boolean values are <code>true</code> and <code>false</code> . These are not case sensitive, and these are the only values that don't use quotation marks. Don't use <code>Yes (y)</code> , <code>No (n)</code> , <code>1</code> , or <code>0</code> for Booleans.
JSON hash	<code>"puppet_enterprise::profile::orchestrator::java_a": { "Xmx": "256m", "Xms": "256m" }</code>
Integer	<code>"puppet_enterprise::profile::console::rbac_session_t": 60</code>

Important: With the exception of Booleans, always use double quotes ("") around parameter values.

Related information

[Configuration file syntax](#) on page 213

Puppet supports two formats for configuration files: valid JSON and Human-Optimized Config Object Notation (HOCON), which is a JSON superset. We've provided these syntax examples to guide you when you're writing configuration files.

Installation parameters

These parameters must be present in the `pe.conf` file to install Puppet Enterprise (PE).

`puppet_enterprise::puppet_master_host`

Specify the FQDN of the node hosting your PE primary server, such as `primary.example.com`.

Default: `%{::trusted.certname}`

Tip: To simplify installation, keep the default value and then provide a console administrator password after you run the installer.

Related information

[PE ACE server configuration](#) on page 603

The PE ACE server is a service that allows for tasks and catalogs to run against remote targets that can't run a Puppet agent, such as network switches and firewalls.

[PE Bolt server configuration](#) on page 602

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`.

Agent platform parameter

When setting up automated provisioning of an installation, you can define this optional parameter in `pe.conf` to specify the agent platforms you want to support in your installation. If your primary server is connected to the internet when you install or upgrade PE, then the packages for the agent platforms you specified in `pe.conf` are automatically downloaded to the primary server and the platform tags are automatically added as `pe_repo::platform::` classes in the **PE Master** node group, so the agent packages are available to install on nodes in your inventory.

`agent_platform`

Define the parameter using an array containing platform tags like `"ubuntu-22.04-amd64"`. You must format the platform tags you include in the array to match the `platform_tag` fact values referenced in `puppet-agent` packages.

Related information

[Installing agents](#) on page 145

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

Database configuration parameters

These parameters and values are supplied for Puppet Enterprise (PE) databases.

 **CAUTION:** Don't change these parameters. This list is provided only for reference purposes.

`puppet_enterprise::activity_database_name`

The activity database name.

Default: `pe-activity`

`puppet_enterprise::activity_database_read_user`

An activity database user that can perform only read functions.

Default: `pe-activity-read`

`puppet_enterprise::activity_database_write_user`

An activity database user that can perform read and write functions.

Default: `pe-activity-write`

`puppet_enterprise::activity_database_superuser`

The activity database superuser.

Default: `pe-activity`

`puppet_enterprise::activity_service_migration_db_user`

An activity service database user used for migrations.

Default: `pe-activity`

`puppet_enterprise::activity_service_regular_db_user`

An activity service database user used for normal operations.

Default: `pe-activity-write`

`puppet_enterprise::classifier_database_name`

The classifier database name.

Default: `pe-classifier`

puppet_enterprise::classifier_database_read_user
A classifier database user that can perform only read functions.
Default: pe-classifier-read

puppet_enterprise::classifier_database_write_user
A classifier database user that can perform read and write functions.
Default: pe-classifier-write

puppet_enterprise::classifier_database_superuser
The classifier database superuser.
pe-classifier

puppet_enterprise::classifier_service_migration_db_user
A classifier service user used for migrations.
Default: pe-classifier

puppet_enterprise::classifier_service_regular_db_user
A classifier service user used for normal operations.
Default: pe-classifier-write

puppet_enterprise::orchestrator_database_name
The orchestrator database name.
Default: pe-orchestrator

puppet_enterprise::orchestrator_database_read_user
An orchestrator database user that can perform only read functions.
Default: pe-orchestrator-read

puppet_enterprise::orchestrator_database_write_user
An orchestrator database user that can perform read and write functions.
Default: pe-orchestrator-write

puppet_enterprise::orchestrator_database_superuser
The orchestrator database superuser.
Default: pe-orchestrator

puppet_enterprise::orchestrator_service_migration_db_user
An orchestrator service user used for migrations.
Default: pe-orchestrator

puppet_enterprise::orchestrator_service_regular_db_user
An orchestrator service user used for normal operations.
Default: pe-orchestrator-write

puppet_enterprise::puppetdb_database_name
The PuppetDB database name.
Default: pe-puppetdb

puppet_enterprise::puppetdb_database_user
The PuppetDB database user.
Default: pe-puppetdb

Tip: If necessary, you can [Change the PuppetDB user password](#) on page 221.

puppet_enterprise::rbac_database_name

The role-based access control (RBAC) database name.

Default: pe-rbac

puppet_enterprise::rbac_database_read_user

An RBAC database user that can perform only read functions.

Default: pe-rbac-read

puppet_enterprise::rbac_database_write_user

An RBAC database user that can perform read and write functions.

Default: pe-rbac-write

puppet_enterprise::rbac_database_superuser

The RBAC database superuser.

Default: pe-rbac

puppet_enterprise::rbac_service_migration_db_user

An RBAC service user used for migrations.

pe-rbac

puppet_enterprise::rbac_service_regular_db_user

An RBAC service user used for normal operations.

Default: pe-rbac-write

External PostgreSQL parameters

These parameters are required to install an external PostgreSQL instance. If necessary, you can add password parameters to standard installations.

puppet_enterprise::database_host

The agent certname of the node hosting the database component.

Important: Don't use an alt name for the `database_host` value.

puppet_enterprise::database_port

The port that the database is running on.

Default: 5432

puppet_enterprise::database_ssl

A Boolean indicating whether SSL authentication is used.

Default: true

Important: Don't use SSL security for unmanaged PostgreSQL installations. Make sure you set `database_ssl` to `false`.

puppet_enterprise::database_cert_auth

A Boolean indicating whether certificate authentication is used.

Default: true

Important: Don't use SSL security for unmanaged PostgreSQL installations. Make sure you set `database_cert_auth` to `false`.

puppet_enterprise::puppetdb_database_password

Specify a password, as a string, for the PuppetDB database user.

For example: mypassword

puppet_enterprise::classifier_database_password

Specify a password, as a string, for the classifier database user.

For example: mypassword

puppet_enterprise::classifier_service_regular_db_user

A database user the classifier service can use for normal operations.

Default: pe-classifier

puppet_enterprise::classifier_service_migration_db_user

A database user the classifier service can use for migrations.

Default: pe-classifier

puppet_enterprise::activity_database_password

Specify a password, as a string, for the activity database user.

For example: mypassword

puppet_enterprise::activity_service_regular_db_user

A database user the activity service can use for normal operations.

Default: pe-activity

puppet_enterprise::activity_service_migration_db_user

A database user the activity service can use for migrations.

Default: pe-activity

puppet_enterprise::rbac_database_password

Specify a password, as a string, for the RBAC database user.

For example: mypassword

puppet_enterprise::rbac_service_regular_db_user

A database user the RBAC service can use for normal operations.

Default: pe-rbac

puppet_enterprise::rbac_service_migration_db_user

A database user the RBAC service can use for migrations.

Default: pe-rbac

puppet_enterprise::orchestrator_database_password

Specify a password, as a string, for the orchestrator database user.

For example: mypassword

puppet_enterprise::orchestrator_service_regular_db_user

A database user the orchestrator service can use for normal operations.

Default: pe-orchestrator

puppet_enterprise::orchestrator_service_migration_db_user

A database user the orchestrator service can use for migrations.

Default: pe-orchestrator

Primary server parameters

Use these parameters to configure and tune the primary server.

pe_install::puppet_master_dnsltnames

An array of strings representing DNS altnames to add to the primary server's SSL certificate.

Default: ["puppet"]

`pe_install::install::classification::pe_node_group_environment`

A string indicating the environment that infrastructure nodes are running in.

Specify this parameter if you moved your primary server and other infrastructure nodes from the default production environment after install. With non-default environments, this setting ensures that your configuration settings are backed up.

Default: `production`

`puppet_enterprise::ip_version`

Accepts either 4 or 6 to specify a preference for IPv4 or IPv6, but this does not restrict the non-preferred option.

The default is 4 (prefer IPv4). You can set it to 6 if you prefer IPv6.

`puppet_enterprise::ipv6_only`

You can set this to `true` to force NGINX to listen only on IPv6.

The default is `false`, which allows both IPv4 and IPv6.

`puppet_enterprise::master::recover_configuration::pe_environment`

A string indicating the environment that infrastructure nodes are running in.

Specify this parameter if you moved your primary server and other infrastructure nodes from the default production environment after installation. With non-default environments, this setting ensures that your configuration settings are backed up.

Default: `production`

`puppet_enterprise::profile::certificate_authority`

An 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`

A Boolean indicating whether to check for updates when the `pe-puppetserver` service restarts.

The default is `true` (check for updates). You can set it to `false` to not check for updates.

`puppet_enterprise::profile::master::code_manager_auto_configure`

Set to `true` to automatically configure the Code Manager service; otherwise, set it to `false`.

`puppet_enterprise::profile::master::r10k_remote`

A string representing the Git URL to be passed to the `r10k.yaml` file, for example:

`git@your.git.server.com:puppet/control.git`

The URL can be any URL supported by r10k and Git.

This parameter is only required if you want r10k configured when you install PE, and you must also specify `puppet_enterprise::profile::master::r10k_private_key`.

`puppet_enterprise::profile::master::r10k_private_key`

A string representing the local file path on the primary server where the SSH private key can be found and used by r10k, for example: `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`

This parameter is only required if you want r10k configured when you install PE, and you must also specify `puppet_enterprise::profile::master::r10k_remote`.

Console and console-services parameters

In the **PE Console** node group, these parameters customize the behavior of the console and the `console-services` service.

You can modify parameters that begin with `puppet_enterprise::profile` in the PE console.

`puppet_enterprise::profile::console::classifier_synchronization_period`

An integer representing, in seconds, the classifier synchronization period. This controls how long the node classifier can take to retrieve classes from the primary server.

Default: 600

`puppet_enterprise::profile::console::ldap_sync_period_seconds`

An integer specifying, in seconds, the interval at which LDAP user details and group membership associations are synchronized.

The default value is 1800 (30 minutes).

This synchronization refreshes the user details and group membership for every LDAP user in the system, regardless of the last time the user logged in. If a user is no longer present in LDAP, all user-group associations are removed from the user and all of the user's known tokens are revoked.

To disable automatic synchronization, set the value to 0 or a negative integer. When disabled, user details and group membership only refresh when the user logs in.

When enabled, various entries are recorded to `console-services.log` that indicate whether the service is enabled and when each synchronization event has completed.

`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.

Default: 10

`puppet_enterprise::profile::console::rbac_password_reset_expiration`

An integer representing the number of hours that `password reset` tokens are valid.

An administrator generates these token for users to reset their passwords.

Default: 24

`puppet_enterprise::profile::console::rbac_session_timeout`

An 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

`puppet_enterprise::profile::console::session_maximum_lifetime`

A string representing how long a console session can last.

The value is formatted as a string consisting of a number and an optional suffix representing a unit of time: `s` (seconds), `m` (minutes), `h` (hours), `d` (days), or `y` (years).

Example: `"1d"` (one day)

If the suffix is omitted, the default unit of time is seconds.

A value of `"0"` sets an unlimited console session time.

To prevent console sessions from expiring before the maximum RBAC token lifetime, set this parameter to `"0"`.

`puppet_enterprise::profile::console::rbac_token_auth_lifetime`

A string representing the default authentication lifetime for a token.

The value is formatted as a string consisting of a number followed by a suffix representing a unit of time: `y` (years), `d` (days), `h` (hours), `m` (minutes), or `s` (seconds).

Important: This value cannot exceed the `rbac_token_maximum_lifetime`.

Default: `"1h"` (one hour)

`puppet_enterprise::profile::console::rbac_token_maximum_lifetime`

A string representing the maximum allowable lifetime for all tokens.

The value is formatted as a string consisting of a number followed by a suffix representing a unit of time: `y` (years), `d` (days), `h` (hours), `m` (minutes), or `s` (seconds).

Default: `10y` (10 years)

puppet_enterprise::profile::console::console_ssl_listen_port

An integer representing the port that the console listens on.

Default: 443

puppet_enterprise::profile::console::ssl_listen_address

A string containing an IP address representing the console's NGINX listen address.

Default: "0.0.0.0"

puppet_enterprise::profile::console::classifier_prune_threshold

An integer representing the number of days to wait before pruning the node classifier database. The node classifier database contains node check-in history if `classifier_node_check_in_storage` is enabled.

Set the value to 0 to never prune the node classifier database.

Default: 7 (days), but only has data to prune if `classifier_node_check_in_storage` is true.

puppet_enterprise::profile::console::classifier_node_check_in_storage

A Boolean specifying whether to create records when nodes check in with the node classifier. These records describe how nodes match the node groups they're classified into.

Set to `true` to enable node check-in storage. Enabling this parameter is required to use [Nodes check-in history endpoints](#) on page 548.

Set to `false` to disable node check-in storage.

Default: `false`

puppet_enterprise::profile::console::display_local_time

A Boolean indicating whether to show timestamps in the local time or UTC.

Set to `true` to display timestamps in local time with hover text showing the equivalent UTC time.

Set to `false` to show timestamps in UTC time with no hover text.

Default: `false`

puppet_enterprise::profile::console::disclaimer_content_path

Specifies the location of the `disclaimer.txt` file containing disclaimer content that can appear on the console login page if you [Create a custom login disclaimer](#) on page 267.

Default: "/etc/puppetlabs/console-services"

Tip: You can also use the RBAC API [Disclaimer endpoints](#) on page 357 to configure the disclaimer without needing to reference a specific file location on disk.

The parameters must be set in Hiera or `pe.conf`, not the console:

puppet_enterprise::api_port

An integer specifying the 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)

For more information, refer to [Node run statuses](#) on page 382.

console_admin_password

The password to log into the console as the admin.

Example: "myconsolepassword"

Default: Specified during installation.

Tip: You can also [Reset the console administrator password](#) on page 266 from the command line.

Related information

[Create a custom login disclaimer](#) on page 267

You can add a custom banner to console login page. For example, you can add a disclaimer about authorized or unauthorized use of private information found in the console.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

Orchestrator and orchestration services parameters

Use these parameters to configure and tune the orchestrator and orchestration services.

`puppet_enterprise::profile::agent::pxp_enabled`

Boolean used to enable or disable the Puppet Execution Protocol (PXP) service.

Set to `true` to enable the PXP service, which is required to use the orchestrator and run Puppet from the console.

Set to `false` to disable the PXP service. If `false`, you can't use the orchestrator or the **Run Puppet** button in the console.

Must be `true` to [Configure PXP agent parameters](#) on page 238.

Default: `true`

`puppet_enterprise::profile::bolt_server::concurrency`

An integer that determines the maximum number of simultaneous task or plan requests the orchestrator can make to `bolt-server`.

This setting only limits task or plan executions on nodes with SSH or WinRM transport methods, because these are the only tasks and plans requiring 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::orchestrator::global_concurrent_compiles`

An integer specifying how many concurrent compile requests can be outstanding to the primary server across all orchestrator jobs.

Default: 8 requests

`puppet_enterprise::profile::orchestrator::job_prune_threshold`

An integer of 2 or greater, which specifies the number of days to retain job reports.

This parameter sets the corresponding parameter `job-prune-days-threshold`.

While `job_prune_threshold` itself has no default value, `job-prune-days-threshold` has a default of 30 (30 days).

`puppet_enterprise::profile::orchestrator::pcp_timeout`

An integer representing how long, in seconds, an agent can spend attempting to connect to a PCP broker during a Puppet run triggered by the orchestrator. If the agent can't connect to the broker in the specified time frame, the Puppet run times out.

Default: 30

`puppet_enterprise::profile::orchestrator::run_service`

A Boolean used to enable (`true`) or disable (`false`) orchestration services.

Default: `true`

puppet_enterprise::profile::orchestrator::task_concurrency

An integer representing the number of simultaneous task or plan actions that can run at the same time. All task and plan actions are limited by this concurrency limit regardless of transport type (WinRM, SSH, PCP).

If a task or plan action runs on multiple nodes, each node consumes one action. For example, if a task needs to run on 300 nodes, and your `task_concurrency` is set to 200, then the task can run on 200 nodes while the remaining 100 nodes wait in queue.

Default: 250 actions

puppet_enterprise::pxp_agent::ping_interval

An integer specifying the frequency, in seconds, that PXP agents ping PCP brokers. If the broker doesn't respond, the agent tries to reconnect.

Default: 120

More information: [Configure PXP agent parameters](#) on page 238

puppet_enterprise::pxp_agent::pxp_logfile

The path, as a string, to the PXP agent log file. This file can be used to debug orchestrator issues.

The default value varies by OS.

- *nix: "/var/log/puppetlabs/pxp-agent/pxp-agent.log"
- Windows: "C:\Program Data\PuppetLabs\pxp-agent\var\log\pxp-agent.log"

More information: [Configure PXP agent parameters](#) on page 238

You might need to configure these parameters depending on your infrastructure. You can always tune them later if you find you need to make adjustments.

puppet_enterprise::profile::orchestrator::allowed_pcp_status_requests

An integer that defines how many times an orchestrator job allows status requests to time out before a job is considered failed. Status requests wait 12 seconds between timeouts, so multiply the value of the `allowed_pcp_status_requests` by 12 to determine how many seconds the orchestrator waits on targets that aren't responding to status requests.

Default: 35 timeouts

puppet_enterprise::profile::orchestrator::java_args

Specifies the [Java heap](#) on page 207 size, which is the amount of JVM memory that each Java process is allowed to request from the OS for orchestration services to use.

The value is formatted as a JSON hash, where the maximum and minimum are usually the same. For example:
`{ "Xmx": "256m", "Xms": "256m" }`

Default: 704 MB

puppet_enterprise::profile::orchestrator::jruby_max_active_instances

An integer that determines the maximum number of JRuby instances that the orchestrator creates to execute plans. Because each plan uses one JRuby to run, this value is effectively the maximum number of concurrent plans. Setting the orchestrator heap size (`java_args`) automatically sets the `jruby_max_active_instances` using the formula `$java_args ÷ 1024`. If the result is less than one, the default is one JRuby instance.

Default: 1 instance

Note: The `jruby_max_active_instances` pool for the orchestrator is separate from the Puppet Server pool. Refer to [JRuby max active instances](#) on page 206 for more information.

puppet_enterprise::profile::plan_executor::versioned_deploys

A Boolean used for [Running plans alongside code deployments](#) on page 651.

Set to `true` to enable versioned deployments of environment code.

Default: false

Important: Setting this to true **disables** the file sync client's locking mechanism that usually enforces a consistent environment state for your plans. This can cause Puppet functions and plans that call other plans to behave unexpectedly if a code deployment occurs while a plan is running.

Related information

[Configuring Puppet orchestrator](#) on page 597

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::profile::master::puppetdb_host`

An array containing a string representing the certname of the node running the PuppetDB service, which is usually the primary server.

Default: ["<PRIMARY_SERVER_CERTNAME>"]

The value is set on the [PE Infrastructure node group](#) on page 456 and inherited by all child infrastructure node groups.

`puppet_enterprise::profile::master::puppetdb_port`

An array containing an integer representing the SSL port that PuppetDB listens on.

Default: [8081]

You might need to change this value if [The PuppetDB default port conflicts with another service](#) on page 863.

The value is set on the [PE Infrastructure node group](#) on page 456 and inherited by all child infrastructure node groups.

`puppet_enterprise::profile::master::puppetdb::report_processor_ensure`

Specifies if you want the primary server to generate *agent run reports* after each Puppet run.

Accepts a string of either "present" or "absent".

Default: "present" (enabled)

Set to "absent" to [Disable agent run reports](#) on page 221.

`puppet_enterprise::profile::puppetdb::node_purge_ttl`

Set the length of time before PE automatically removes deactivated or expired nodes, along with their facts, catalogs, and reports, from PuppetDB.

Specify a string representing an amount of time. For example, "14d" sets the retention time to 14 days.

Default: "14d"

For more information, refer to [Set the deactivated node retention time](#) on page 221.

`puppet_enterprise::profile::puppetdb::command_processing_threads`

Integer representing how many command processing threads PuppetDB uses to sort incoming data. Each thread can process one command at a time.

If the PuppetDB service runs on compilers, the default value is 25% of the number of cores in your system. Otherwise, the default value is half the number of cores in your system. The minimum is 1.

For more information, refer to [PuppetDB command processing threads](#) on page 209.

Related information

[Database configuration parameters](#) on page 130

These parameters and values are supplied for Puppet Enterprise (PE) databases.

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" }.

Related information

[Java heap](#) on page 207

The `java_args` settings specify *heap size*, which is the amount of memory that each Java process can request from the operating system. You can specify a heap size for each PE service that uses Java, including Puppet Server, PuppetDB, the console, and the orchestrator

Install PE using PIM

Puppet Installation Manager (PIM) supports the deployment of standard, large, and extra-large PE architectures. For an interactive experience, choose the guided installation and follow the step-by-step process in your terminal to configure and install the PE infrastructure you require. Alternatively, if you do not require guidance, you can create a JSON file containing your custom installation parameters, and run the installation from the command line.

Regardless of the installation process you choose, you can use PIM on a jump host to install PE infrastructure components on remote nodes that run a supported PE operating system. Alternatively, you can install PE locally by using PIM on a machine running a supported PE operating system. In this scenario, if you require additional infrastructure nodes to host PE components, your local machine can serve as a jump host.

PE infrastructure nodes are the hosts where PE components are installed. The following table lists the infrastructure nodes you can include in your installation when you use PIM to install PE.

PE infrastructure node	Description
Primary server (required)	Essential for a PE installation. Can host all components and services for smaller scale environments that include up to 2,000 nodes.
Primary server replica (optional)	To set up disaster recovery, install a replica of the primary server. If your primary server fails, the replica takes over to continue critical operations.
Database server	In an extra-large PE installation, a dedicated database server hosts a PostgreSQL instance containing the PuppetDB database.
Database server replica (optional)	Provides backup support during failovers.

PE infrastructure node	Description
Compilers	Compilers process Puppet code and convert it into catalogs that can be applied to agent nodes. The primary server can handle requests and compile catalogs for up to 2,000 agent nodes. In large and extra-large PE installations, dedicated compiler nodes help accelerate catalog compilation.

Related information

[Supported architectures](#) on page 92

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. When you install PE using the PE installer tarball, you begin with the standard configuration, and can then scale up by adding additional infrastructure components as needed. Alternatively, by using Puppet Installation Manager (beta) to install PE, you can start out with a standard, large, or extra-large configuration.

[What gets installed and where?](#) on page 116

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.

[Supported operating systems](#) on page 98

Puppet Enterprise supports various operating systems depending on the role a machine assumes in your infrastructure.

Install PE using the guided process

For an interactive experience, use the guided installation process. Based on information you provide about your environment and requirements, PIM automatically configures your PE installation.

Before you begin

- Ensure that you have the required access to the nodes where you want to install PE infrastructure.
 - To install the primary server locally on the machine where PIM is running, you must log in as the root user.
 - To install PE components on remote nodes, the machine running PIM must have SSH access to the target nodes, and the user executing the installation must have superuser privileges for those nodes.
- Ensure that Puppet is not already installed on any of the nodes where you want to install PE infrastructure.
- Check system requirements:
 - [Hardware requirements](#)
 - [Supported operating systems](#)
 - [Supported browsers](#)
 - [System configuration](#)

Important: Security-Enhanced Linux (SELinux) is enabled and enforced by default on Red Hat Enterprise Linux 9 (RHEL 9) operating systems. In order to use PIM, users must provide permission for PIM binary.

To install PE by using the PIM guided process:

1. Download PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Start the guided installation process.

In your terminal, navigate to the `pim` directory and run the following command:

```
./pim wizard
```

3. Follow the guided steps in your terminal to complete the installation.

If you require additional guidance during the installation process, you can view help content by pressing `Ctrl+H`.

Install PE with your defined parameters

If you know which PE infrastructure components you want to install and you do not require guidance, you can specify your installation parameters in a JSON file. Then use PIM to start the installation by running a single command.

Before you begin

- Ensure that you have the required access to the nodes where you want to install PE infrastructure.
 - To install the primary server locally on the machine where PIM is running, you must log in as the root user.
 - To install PE components on remote nodes, the machine running PIM must have SSH access to the target nodes, and the user executing the installation must have superuser privileges for those nodes. You can configure SSH, or use the `-b` flag to pass the SSH key or SSH credentials when you run the installation command.
- Ensure that Puppet is not already installed on any of the nodes where you want to install PE infrastructure.
- Check system requirements:
 - [Hardware requirements](#)
 - [Supported operating systems](#)
 - [Supported browsers](#)
 - [System configuration](#)

To install PE from the PIM command line:

1. Download PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Create a JSON file specifying the installation parameters you require.

For examples illustrating the JSON properties required for different PE architectures, see [Creating an installation parameters file](#).

3. Start the installation.

In your terminal, navigate to the `pim` directory and run one of the following commands, replacing `parameters.json` with the actual file name (including the file path, if necessary):

- To run the installation without debugging and without configuring SSH, use a command like the following example:

```
./pim install parameters.json
```

- To enable debug logging, add `-d` or `--debug`. For example:

```
./pim install parameters.json --debug
```

- To pass an SSH key or SSH credentials for accessing remote nodes, use the `-b` flag with the installation command as shown in the following examples:

```
./pim install -b user=root -b private-key=~/ssh/ssh_key params.json
```

```
./pim install -b user=root -b password=ssh_password params.json
```

4. Follow the CLI prompts to complete the installation process.

Note: PIM uses the Puppet Enterprise Administration Module (PEADM), which depends on Puppet Bolt, a tool for automating Puppet infrastructure maintenance tasks. When you run the `./pim install` command, PIM checks whether Bolt is present and, if necessary, provides the option to install Bolt.

Creating an installation parameters file

To install PE from the Puppet Installation Manager (PIM) command line, you must create a JSON file containing your installation parameters and pass that file with the `install` command. The JSON file defines your installation architecture, including the option for disaster recovery.

Important: Creating a JSON file containing installation parameters is not required if you use the guided installation process. With the guided process, PIM automatically configures your installation based on the information you provide about your environment and requirements.

Installation configuration examples

The following examples illustrate how to structure the JSON file for different PE configurations.

Installation parameters for an extra-large architecture with disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "replica_host": "pe-xl-core-2.lab1.puppet.vm",
  "replica_postgresql_host": "pe-xl-core-3.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}
```

Installation parameters for an extra-large architecture without disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}
```

Installation parameters for a large architecture with disaster recovery

```
{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
  "replica_host": "pe-l-core-2.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
  ],
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}
```

Installation parameters for a large architecture without disaster recovery

```
{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
```

```

"compiler_hosts": [
  "pe-1-compiler-0.lab1.puppet.vm",
  "pe-1-compiler-1.lab1.puppet.vm"
],
"console_password": "puppetlabs",
"dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
"version": "2023.6.0"
}

```

Installation parameters for a standard architecture with disaster recovery

```

{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "replica_host": "pe-core-2.lab1.puppet.vm",
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Installation parameters for a standard architecture without disaster recovery

```

{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "console_password": "puppetlabs",
  "dns_alt_names": [ "puppet", "puppet.lab1.puppet.vm" ],
  "version": "2023.6.0"
}

```

Purchasing and installing a license key

A Puppet Enterprise (PE) license includes access to Security Compliance Management (formerly Comply) and Continuous Delivery, both of which can be installed and used after you have installed PE. To unlock additional premium features including Security Compliance Enforcement (formerly CEM) and advanced Impact Analysis capabilities in Continuous Delivery, you can [contact our sales team](#).

Your license must support the number of nodes that you want to manage with PE.

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 437

Your *node count* is the number of nodes in your inventory. Your Puppet Enterprise (PE) 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 433

Purging a node removes it from your inventory so it is no longer managed by Puppet Enterprise (PE) and allows you to use the node's license on another node.

Install a license key

Install the `license.key` file to upgrade from a test installation to an active installation.

1. Install the license key by copying the file to `/etc/puppetlabs/license.key` on the primary server node.
2. Verify that Puppet has permission to read the license key by checking its ownership and permissions: `ls -la /etc/puppetlabs/license.key`
3. If the ownership is not `root` and permissions are not `-rw-r--r--` (octal 644), set them:

```
sudo chown root:root /etc/puppetlabs/license.key
sudo chmod 644 /etc/puppetlabs/license.key
```

View license details for your environment

Check the number of active nodes in your deployment, the number of licensed nodes you purchased, and the expiration date for your license.

Procedure

- In the console, click **License**.

The **License** page opens with information on your licensed nodes, bursting limit, and subscription expiration date. Any license warnings that appear in the console navigation are explained here. For example, if your license is expired or out of compliance.

Related information

[Usage endpoints](#) on page 753

Use the `usage` endpoint to view details about your deployment's active nodes.

[Remove agent nodes](#) on page 433

Purging a node removes it from your inventory so it is no longer managed by Puppet Enterprise (PE) and allows you to use the node's license on another node.

Installing agents

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

There are multiple ways to install agents. We recommend using the install script or installing agents from the console, and we have provided instructions for other cases, such as non-root agents, offline installation, and manually-transferred certificates. After installing agents, you must accept their certificate signing requests (CSRs).

You usually install agents from the PE package management repository on your primary server, which is created when you install your primary server. This repository serves packages over HTTPS using the same port as the primary server (port 8140). This means agent nodes don't require you to open any ports other than the one they already use to communicate with the primary server.

You can find agent packages on the primary server at `/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 runs on CentOS 7, in the agent packages directory there is a directory named `e1-7-x86_64`. This directory contains multiple subdirectories with the packages needed to install an agent.

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.

Tip:

After installing agents, you can edit node configuration settings in each node's `puppet.conf` file at `/etc/puppetlabs/puppet/puppet.conf`. You can edit this file directly or use the `puppet config set` sub-command.

For example, to point an agent at a primary server called `primary.example.com`, run `puppet config set server primary.example.com`. This command adds `server = primary.example.com` to the main section of the node's `puppet.conf` file.

The [Puppet Configuration Reference](#) explains the configuration settings you can specify in `puppet.conf`.

All agent installation instructions assume your nodes use [Supported operating systems](#) on page 98.

- [Install agents with the install script](#) on page 147

You can use the install script for *nix, Windows, and macOS nodes. The install script installs and configures the agent on target nodes using installation packages from the Puppet Enterprise (PE) package management repo.

- [Install agents from the console](#) on page 150

You can use the Puppet Enterprise (PE) console to install agents in *nix, macOS, and Windows nodes.

- [Install *nix agents](#) on page 151

You can install agents on *nix nodes with the install script, from the Puppet Enterprise (PE) console, with PE package management, your own package management, with or without internet access, and more.

- [Install Windows agents](#) on page 154

There are many ways you can install agents on Windows nodes, including PowerShell scripts, the Puppet Enterprise (PE) console, the MSI installer, and the `msiexec` command.

- [Install macOS agents](#) on page 160

On macOS, agents have core Puppet functionality and platform-specific capabilities like package installation, LaunchD service management, System Profiler facts inventory, and directory services integration. You can install agents on macOS nodes with the install script, from the Puppet Enterprise (PE) console, from Finder, and more.

- [Install non-root agents](#) on page 161

You can configure non-root agents on *nix and Windows nodes. Running agents without root privileges allows teams to perform some, but not all, administrative actions in Puppet Enterprise (PE) that would otherwise require root privileges.

- [Managing certificate signing requests](#) on page 164

When you install a Puppet agent on a node, the agent must submit a certificate signing request (CSR) to the primary server, and you must accept the CSR to add the node to your Puppet Enterprise (PE) inventory. Accepting the CSR 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 PE console.

Related information

[Upgrading agents](#) on page 196

Upgrade your agents as new versions of Puppet Enterprise (PE) become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can use a script to upgrade individual nodes.

[Set a proxy for agent traffic](#) on page 229

General proxy settings in an agent node's `puppet.conf` file are used to manage HTTP connections directly initiated by the agent node.

[Adding and removing agent nodes](#) on page 432

You can add nodes you want to manage with Puppet Enterprise (PE) and remove nodes you no longer need.

[FIPS 140-2 enabled PE](#) on page 17

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.

[Agent platform parameter](#) on page 130

When setting up automated provisioning of an installation, you can define this optional parameter in `pe.conf` to specify the agent platforms you want to support in your installation. If your primary server is connected to the internet when you install or upgrade PE, then the packages for the agent platforms you specified in `pe.conf` are automatically downloaded to the primary server and the platform tags are automatically added as `pe_repo::platform::` classes in the **PE Master** node group, so the agent packages are available to install on nodes in your inventory.

[Uninstall agents](#) on page 177

You can remove the `puppet-agent` package from nodes that you no longer want Puppet Enterprise (PE) to manage.

Install agents with the install script

You can use the install script for *nix, Windows, and macOS nodes. The install script installs and configures the agent on target nodes using installation packages from the Puppet Enterprise (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 declare on the primary server (in the console at **Node Groups > PE Master > Classes**).
- Downloads a plug-in tarball from the primary server. This feature is controlled by the `pe_repo::enable_bulk_pluginsync` and `pe_repo::enable_windows_bulk_pluginsync` settings, which are set to `true` (enabled) by default.

Note: Depending on how many modules you have installed, bulk plug-in sync can improve agent installation speed. However, if your primary server runs on a different platform than your agent nodes, bulk plug-in sync might be less beneficial. The plug-in tarball is based on the plug-ins running on the primary server's agent, which might not match the plug-ins required for agents on other platforms.

- Creates a basic `puppet.conf` file containing the node's configuration settings. This file is stored at `/etc/puppetlabs/puppet/puppet.conf`.
- Kicks off a Puppet run.

Use the install script

Before you begin: The agent node must have an internet connection to download the agent installer packages and plug-ins.

If you're installing an agent with a different OS than your primary server, you must declare the corresponding `pe_repo` class on the primary server, such as `pe_repo::platform::el_7_x86_64`. Declare these classes in the console at **Node Groups > PE Master > Classes**.

If your primary server is airgapped or uses a proxy server to access the internet, before installing agents, you must specify `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` in the **PE Master** node group's `pe_repo` class. For information about how to download agent installation packages through a proxy, see [Configure proxies](#) on page 228.

1. In the PE console, go to **Nodes > Add nodes > Install agents**.

2. Under **Manual installation**, copy the command corresponding with your node's OS. You can use the ***nix nodes** script for *nix and macOS nodes.
3. Launch the install script by running the command you copied. For Windows nodes, run the command in an administrative PowerShell window.

Remember: If the install script can't find agent packages corresponding to the agent's OS, it fails with an error telling you which `pe_repo::platform` class you need to declare on the primary server (at **Node Groups > PE Master > Classes**).

4. Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
5. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Customize the install script

If necessary, you can use these options to modify the install script to define specific agent configuration settings, CSR attributes, or MSI properties. You can also control whether the Puppet service is running or enabled after agent installation.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

`puppet.conf` settings

You can use the install script to specify agent configuration settings in the node's `puppet.conf` file, which is generated by the install script.

The Puppet [Configuration Reference](#) explains the configuration settings you can specify in `puppet.conf` and provides tips for successfully defining settings. Some commonly-specified settings include:

- `server`
- `certname`
- `environment`
- `splay`
- `splaylimit`
- `noop`

You can specify an unlimited number of settings in any order. In the install script command, use the `section:key=value` pattern to define each setting and leave one space between settings. In the *nix install script command, use `-s` to introduce the assortment of settings.

For example, for an Enterprise Linux system with a proxy between the agent and primary server, you can specify the `http_proxy_host` setting by adding the following code to the install script command:

```
-s agent:http_proxy_host=<PROXY_FQDN>
```

As another example, the following code specifies the `splay`, `certname`, and `environment` settings in the `main` and `agent` sections of the `puppet.conf` file:

```
main:certname=node1.company.com \
agent:splay=true \
agent:environment=development
```

The `puppet.conf` file resulting from this code contains:

```
[main]
certname = node1.corp.net
```

```
[agent]
splay = true
environment = development
```

Tip: You can also edit node configuration settings after running the script by editing the `puppet.conf` file directly (at `/etc/puppetlabs/puppet/puppet.conf`) or using the `puppet config set` sub-command.

For example, to point an agent at a primary server called `primary.example.com`, run `puppet config set server primary.example.com`. This command adds `server = primary.example.com` to the `[main]` section of the node's `puppet.conf` file.

CSR attribute settings

Certificate signing request attribute settings are added to the node's `puppet.conf` file and are included in the `custom_attributes` and `extension_requests` sections of the `csr_attributes.yaml` file. The Puppet [csr_attributes.yaml: Certificate extensions](#) reference provides details about these settings.

You can specify an unlimited number of settings in any order. In the install script command, use the `section:key=value` pattern to define each setting and leave one space between settings. In the *nix install script command, use `-s` to introduce the assortment of settings.

For example, these commands specify agent and certificate signing settings:

```
-s main:certname=<CERTNAME_OTHER_THAN_FQDN> \
custom_attributes:challengePassword=<PASSWORD_FOR_AUTOSIGNER_SCRIPT> \
extension_requests:pp_role=<PUPPET_NODE_ROLE>
```

The above code adds the `main:certname` setting to the `puppet.conf` file and a `csr_attributes.yaml` file containing:

```
---
custom_attributes:
  challengePassword: <PASSWORD_FOR_AUTOSIGNER_SCRIPT>
extension_requests:
  pp_role: <PUPPET_NODE_ROLE>
```

Tip: If you can't run the install script, you can set CSR attributes by manually creating a `csr_attributes.yaml` file in the Puppet `confdir` (at `C:\ProgramData\PuppetLabs\puppet\etc\csr_attributes.yaml`) prior to installing the Puppet agent package with another agent installation method.

MSI properties (Windows only)

For the Windows install script, you can set these MSI properties with or without additional agent configuration settings.

MSI Property	PowerShell flag
INSTALLDIR	<code>-InstallDir</code>
PUPPET_AGENT_ACCOUNT_USER	<code>-PuppetAgentAccountUser</code>
PUPPET_AGENT_ACCOUNT_PASSWORD	<code>-PuppetAgentAccountPassword</code>
PUPPET_AGENT_ACCOUNT_DOMAIN	<code>-PuppetAgentAccountDomain</code>

For example, adding this code to the Windows install script runs the Puppet service as `pup_adm` with the defined password:

```
-PuppetAgentAccountUser 'pup_adm' -PuppetAgentAccountPassword '<PASSWORD>' -  
PuppetAgentAccountDomain '<DOMAIN>'
```

Important: If you specify `PUPPET_AGENT_ACCOUNT_USER`, you must also specify `PUPPET_AGENT_ACCOUNT_PASSWORD` and `PUPPET_AGENT_ACCOUNT_DOMAIN` unless the node is under a gMSA.

For gMSAs, you must specify `PUPPET_AGENT_ACCOUNT_USER` (the user for the gMSA) and `PUPPET_AGENT_ACCOUNT_DOMAIN`. Do not specify `PUPPET_AGENT_ACCOUNT_PASSWORD`.

If you need to specify additional MSI properties, you might need to [Install Windows agents with the .msi package](#) on page 155.

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 the Puppet run, or you're using a provisioning system that requires non-default behavior, you can control whether the service is running or enabled.

- `ensure` controls whether the Puppet service is running.
 - Accepts values of `running` or `stopped`.
 - *nix format: `--puppet-service-ensure <VALUE>`
 - Windows format: `-PuppetServiceEnsure <VALUE>`
- `enable` controls whether the Puppet service is enabled.
 - Accepts values of `true`, `false`, `mask`, or `manual` (Windows only).
 - *nix format: `--puppet-service-enable <VALUE>`
 - Windows format: `-PuppetServiceEnable <VALUE>`

For example, to stop the Puppet service, ensure it doesn't boot after installation, and prevent a Puppet run from occurring after the agent is installed, include these settings in the *nix install script command:

```
-s --puppet-service-ensure stopped --puppet-service-enable false
```

To do this in the Windows install script command, include:

```
-PuppetServiceEnsure stopped -PuppetServiceEnable false
```

Install agents from the console

You can use the Puppet Enterprise (PE) console to install agents in *nix, macOS, and Windows nodes.

Before you begin

If you're installing an agent with a different OS than your primary server, you must declare the corresponding `pe_repo` class on the primary server, such as `pe_repo::platform::el_7_x86_64`. Declare these classes in the console at **Node Groups > PE Master > Classes**.

You must have permission to run the `pe_bootstrap` task to install agents on nodes. The `pe_bootstrap::linux` task is for *nix and macOS targets, while the `pe_bootstrap::windows` task is for Windows targets.

Refer to the **Limitations** section of the [pe_bootstrap task's Forge page](#) for platform, PowerShell, and Microsoft .NET Framework requirements.

- In the PE console, go to **Nodes > Add nodes > Install agents**.

Tip: These steps use the default `pe_bootstrap` task to immediately install agents using one transport method (SSH or WinRM). If you want to schedule the installation, use SSH and WinRM concurrently, or specify task parameters (such as a custom certname or other parameters described on the [pe_bootstrap task's Forge page](#)), click **Advanced install**. Make sure you [accept the CSRs](#) after installing agents.

- Select a transport method to remotely install the agent on the target node.
 - SSH** for *nix and macOS
 - WinRM** for Windows

- Enter the target host names and the credentials required to access them. You can specify multiple targets, but only one set of credentials.

Important: If you use an SSH key, include the begin and end tags.

- Click **Add nodes** to install agents on the specified nodes. You can click **Installation job started** to view the task's job details.

Agents are installed on the target nodes and then they automatically submit certificate signing requests (CSRs) to the primary server.

- After installing agents, CSRs are added to the **Unsigned certificates** list in the console. You must accept the certificates, as explained in [Managing certificate signing requests](#) on page 164.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Install *nix agents

You can install agents on *nix nodes with the install script, from the Puppet Enterprise (PE) console, with PE package management, your own package management, with or without internet access, and more.

We recommend you [Install agents with the install script](#) on page 147 or [Install agents from the console](#) on page 150 whenever possible, and we've described other cases here for your reference. For non-root agents, refer to [Install non-root *nix agents](#) on page 162.

You must enable TLSv1 to install agents on these platforms:

- AIX
- Solaris 11

Related information

[Enable TLSv1](#) on page 845

To comply with security regulations, TLSv1 and TLSv1.1 are disabled by default in 2021.7.z versions of Puppet Enterprise (PE).

Install *nix agents with PE package management

Puppet Enterprise (PE) provides its own package management to help you install agents on *nix and macOS nodes. You can use this process with or without internet access.

Before you begin

If you're installing an agent with a different OS than your primary server, you must declare the corresponding `pe_repo` class on the primary server, such as `pe_repo::platform::el_7_x86_64`. You can declare these classes in the console at **Node Groups > PE Master > Classes**.

If the primary server does not have internet access, [download](#) the appropriate agent tarball, and copy the agent tarball to this location on the primary server:

```
/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-<AGENT_VERSION>
```

For example, the directory for agent version 7.26.0 is:

```
/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-7.26.0/
```

These commands modify the standard agent install script for specific platforms or airgapped environments. If you do not have a specific need for these commands, we recommend you [Install agents with the install script](#) on page 147.

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. SSH into the node where you want to install the agent and run the command appropriate to your environment:

- curl:

```
uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'
curl -k "$uri" | sudo bash
```

- wget:

```
wget -O - -q --no-check-certificate https://<PRIMARY_HOSTNAME>:8140/
packages/current/install.bash | sudo bash
```

- Solaris 11:

```
sudo export PATH=$PATH:/opt/sfw/bin
wget -O - -q --no-check-certificate --secure-protocol=TLSv1 https://
<PRIMARY_HOSTNAME>:8140/packages/current/install.bash | bash
```

2. Run `puppet agent -t` to add the node to the node inventory and generate the CSR.

3. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

[Using example commands](#) on page 28

These guidelines can help you understand and customize the example commands you'll find in the Puppet Enterprise (PE) docs.

[Commands with elevated privileges](#) on page 30

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

Install *nix agents with your own package management

You can use your own package management tools, instead of Puppet Enterprise (PE) package management, to install agents. You can use this method with or without internet access.

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 (such as YUM or APT) to point to that repo.

- Install the agent using the command appropriate to your environment:

- YUM:

```
sudo yum install puppet-agent
```

- APT:

```
sudo apt-get install puppet-agent
```

In offline environments, you might need to disable the PE-hosted package management repo if the installer gets stuck trying to connect to the primary server. To do this, in the PE console, go to **Node groups > PE Infrastructure > PE Master**. On the **Classes** tab, find the `pe_repo::platform` class corresponding with your node's platform, click **Remove this class**, and commit changes.

- Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
- Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Install *nix agents using a manually-transferred certificate

If you can't, or don't, use `-k` or `--insecure` to trust the primary server during agent installation, you can manually copy the primary server CA certificate to any *nix machines you want to install agents on, and then run a variation of the agent install script against that cert.

For general information about forming curl commands and authentication in commands, go to [Using example commands](#) on page 28.

- On the machine where you want to install the agent, create this directory: `/etc/puppetlabs/puppet/ssl/certs`
- On the primary server, navigate to `/etc/puppetlabs/puppet/ssl/certs/` and copy `ca.pem` to the `certs` directory you created on the agent node.
- On the agent node, verify file permissions by running:

```
chmod 444 /etc/puppetlabs/puppet/ssl/certs/ca.pem
```

- Run the agent install script command, using the `--cacert` flag to point to the cert, such as:

```
cacert='/etc/puppetlabs/puppet/ssl/certs/ca.pem'
uri='https://<PRIMARY_HOSTNAME>:8140/packages/current/install.bash'

curl --cacert "$cacert" "$uri" | sudo bash
```

For more information about the agent install script, go to [Install agents with the install script](#) on page 147.

- Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
- Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Install *nix agents from compilers using your own package management

If your infrastructure relies on compilers to install agents, you don't have to copy the agent package to each compiler. Instead, you can use the console to specify a path to the agent package on your package management server.

Before you begin

[Download](#) the appropriate agent tarball.

- Add the agent package to your own package management and distribution system.
- Set the `base_path` parameter of the `pe_repo` class to point to your package management server.
 - In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Master** group.
 - On the **Classes** tab, find the `pe_repo` class, and set the `base_path` parameter to your package management server's FQDN.
 - Click **Add parameter** and commit changes.

- Follow the steps to [Install *nix agents with your own package management](#) on page 152.

Install Windows agents

There are many ways you can install agents on Windows nodes, including PowerShell scripts, the Puppet Enterprise (PE) console, the MSI installer, and the `msiexec` command.

We recommend you [Install agents with the install script](#) on page 147 or [Install agents from the console](#) on page 150 whenever possible, and we've described other cases here for your reference. For non-root agents, refer to [Install non-root Windows agents](#) on page 163.

Install Windows agents with PE package management

Puppet Enterprise (PE) provides its own package management to help you install agents on Windows nodes. 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 tarball and save it to the appropriate agent package directory on your primary server.

- For 32-bit systems, save the tarball at `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/`
- For 64-bit systems, save the tarball at `/opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/`

These commands modify the standard agent install script for specific platforms or airgapped environments. If you do not have a specific need for these commands, we recommend you [Install agents with the install script](#) on page 147.

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.

- On the agent node, open an administrative PowerShell window, and run the appropriate agent install script command:

For Microsoft Windows Server 2008r2:

```
[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
```

For all other Windows platforms:

```
[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
```

After running the install script, the following output indicates the agent was installed successfully:

```
Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'  
service { 'puppet':  
  ensure => 'running',  
  enable => 'true',  
}
```

- Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
- Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

[Configure proxies](#) on page 228

If you have components with limited (or no) internet access, you can configure proxies at various points in your infrastructure, depending on your connectivity limitations.

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 any Windows machines you want to install agents on, and then run a variation of the agent install script against that cert.

1. Transfer the CA certificate:

- On the machine where you want to install the agent, create this directory: C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\
- On the primary server, navigate to: /etc/puppetlabs/puppet/ssl/certs/
- Copy ca.pem to the certs directory you created on the agent node.

2. Transfer the agent install script:

- On the primary server, navigate to: /opt/puppetlabs/server/data/packages/public/
- Copy install.ps1 to any accessible local directory on the agent node.

3. In an administrative PowerShell window, run the install script with the -UsePuppetCA flag: .\install.ps1 -UsePuppetCA

4. Run puppet agent -t to add the node to the node inventory and generate the CSR.

5. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Install Windows agents with the .msi package

You can use the Windows MSI installer or the msieexec command to install the agent .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 appropriate agent .msi package.

To install agents on Windows nodes without internet access, save the .msi package to the appropriate agent package directory:

- For 32-bit systems, save the package at /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-i386-<AGENT_VERSION>/
- For 64-bit systems, save the package at /opt/puppetlabs/server/data/packages/public/<PE_VERSION>/windows-x86_64-<AGENT_VERSION>/

Related information

[Configure proxies](#) on page 228

If you have components with limited (or no) internet access, you can configure proxies at various points in your infrastructure, depending on your connectivity limitations.

Install Windows agents with the MSI installer

Use the MSI installer for an automated installation process. The installer can configure puppet.conf, configure CSR attributes, and connect the agent to your primary server.

- Run the MSI installer as administrator.
- When prompted, provide your primary server's hostname, for example puppet.company.com.
- Once the agent is installed, you must accept the node's CSR as explained in [Managing certificate signing requests](#) on page 164.

Install Windows agents using `msiexec` from the command line

You can install the `.msi` package manually from the command line if you need to customize `puppet.conf`, CSR attributes, or certain agent properties.

If you [Install agents with the install script](#) on page 147 (with PowerShell), you can [Customize the install script](#) on page 148 by specifying CSR attribute settings and some MSI properties. The `msiexec` command does not require PowerShell and allows you to specify more MSI properties.

1. Identify the [MSI properties](#) on page 156 you want to include in the `msiexec` command and prepare the syntax for those properties.
2. If you need to set CSR attributes, create a `csr_attributes.yaml` file in the Puppet `confdir` (at `C:\ProgramData\PuppetLabs\puppet\etc\csr_attributes.yaml`) prior to installing the Puppet agent package. [Customize the install script](#) on page 148 explains how to specify CSR attribute settings.
3. To log installation progress to an `install.txt` log file, include `/l*v install.txt` in your `msiexec` command.
4. On the command line of the node where you want to install the agent, run your `msiexec` command.

The basic command is:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi
```

Your command likely includes additional arguments, such as `/l*v`, `PUPPET_AGENT_CERTNAME`, or any other valid [MSI properties](#) on page 156. For example, this `msiexec` command installs the agent with a primary server located at `puppet.acme.com`:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi PUPPET_SERVER=puppet.acme.com
```

This `msiexec` command installs the agent to a domain user account called `bob` on the `ExampleCorp` domain with the account password of `password`:

```
msiexec /qn /norestart /i <PACKAGE_NAME>.msi
PUPPET_AGENT_ACCOUNT_DOMAIN=ExampleCorp PUPPET_AGENT_ACCOUNT_USER=bob
PUPPET_AGENT_ACCOUNT_PASSWORD=password
```

5. Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
6. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

MSI properties

You can use these MSI properties if you install Windows agents with the `msiexec` command.

Important: The following MSI properties define `puppet.conf` settings:

- `PUPPET_SERVER` corresponds with `server`
- `PUPPET_CA_SERVER` corresponds with `ca_server`
- `PUPPET_AGENT_CERTNAME` corresponds with `certname`
- `PUPPET_AGENT_ENVIRONMENT` corresponds with `environment`

If you use `msiexec` to specify a non-default value for these properties, the installer replaces the default value in `puppet.conf` and re-uses your specified value at upgrade. Therefore, if you need to change these properties after setting them with `msiexec`, don't change them directly in `puppet.conf`; instead, you need to re-run the installer and set a new value.

[Customize the install script](#) on page 148 provides more details on `puppet.conf` settings.

Property	Definition	Default value
INSTALLDIR	Location to install Puppet and its dependencies.	For 32-bit systems: C:\Program Files\Puppet Labs\Puppet For 64-bit systems: C:\Program Files \Puppet Labs\Puppet
PUPPET_SERVER	Hostname where the primary server can be reached.	puppet
PUPPET_CA_SERVER	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.	Value of PUPPET_SERVER
PUPPET_AGENT_CERTNAME	The agent node's certificate name and the name it uses when requesting catalogs.	Value of facter fqdn
Important: Only use lowercase letters, numbers, periods, underscores, and dashes.		
PUPPET_AGENT_ENVIRONMENT	The agent node's environment.	production
Important: If the node already has a puppet.conf file containing a value for the environment variable, specifying it during installation doesn't override that value.		
PUPPET_AGENT_STARTUP_MODE	Whether and how the agent service is allowed to run. Allowed values are: <ul style="list-style-type: none"> Automatic: The agent service runs when Windows starts and remains running in the background. Manual: The agent service can be started in the services console or with net start on the command line. Disabled: The agent service is installed but disabled. You must change its startup type in the services console before you can start the service. 	Automatic

Property	Definition	Default value
PUPPET_AGENT_ACCOUNT_USER	<p>The Windows user account the agent service uses.</p> <p>Use this property when 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:</p> <ul style="list-style-type: none"> • Allows domain users even if they have not accessed the machine before. • Grants Logon as Service to the user. • Add the user to the Administrators group, if the user isn't already a local administrator. <p>Important: If you specify this property, you must also specify PUPPET_AGENT_ACCOUNT_PASSWORD and PUPPET_AGENT_ACCOUNT_DOMAIN unless the node is under a gMSA.</p> <p>For gMSAs, you must also specify PUPPET_AGENT_ACCOUNT_DOMAIN, but do not specify PUPPET_AGENT_ACCOUNT_PASSWORD.</p>	LocalSystem

Property	Definition	Default value
PUPPET_AGENT_ACCOUNT_PASSWORD	Predefined password for the agent's user account. Do not specify this property for nodes running under gMSAs.	No value
PUPPET_AGENT_ACCOUNT_DOMAIN	Domain of the agent's user account.	.
REINSTALLMODE	A default MSI property that controls file copy behavior during installation. Important: If you need to downgrade agents, use REINSTALLMODE=amus when calling msieexec.exe at the command line to prevent removing required files.	From puppet-agent version 1.10.10 and 5.3.4: amus Prior releases: omus

About Windows agents

Windows nodes can fetch configurations from the primary server and apply manifests locally, and respond to orchestration commands.

After installing a Windows node, the **Start Menu** contains a **Puppet** folder with shortcuts for running the agent manually, running Facter, and opening a command prompt to use Puppet tools.

Remember: You must run Puppet with [elevated privileges](#). Select **Run as administrator** when opening the command prompt.

The agent runs as a Windows service. By default, the agent fetches and applies configurations every 30 minutes. The agent service can be started and stopped independently using either the service control manager UI or the command line `sc .exe` utility.

Puppet is automatically added to the machine's PATH environment variable, so you can open any command line and run `puppet`, `facter` and the other batch files that are in the Puppet installation's `bin` directory. Items necessary for the Puppet environment are also added to the shell, but only for the duration of each command's execution.

The installer includes Ruby, Ruby gems, and Facter. If you have existing copies of these applications, such as Ruby, they aren't affected by the re-distributed version included with Puppet.

Program files 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 files directory contains these subdirectories:

Subdirectory	Contents
bin	Scripts for running Puppet and Facter
facter	Facter source
hiera	Hiera source
misc	Resources

Subdirectory	Contents
puppet	Puppet source
service	Code to run the agent as a service
sys	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:

- `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, the `data` directory is located in the `COMMON_APPDATA.aspx` directory. This directory is typically located at `C:\ProgramData\PuppetLabs\`. Because the `COMMON_APPDATA.aspx` 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 directory as its `data` directory, which can result in unexpected settings. We recommend always running Puppet with elevated privileges, unless otherwise specified for specific scenarios.

Install macOS agents

On macOS, agents have core Puppet functionality and platform-specific capabilities like package installation, LaunchD service management, System Profiler facts inventory, and directory services integration. You can install agents on macOS nodes with the `install` script, from the Puppet Enterprise (PE) console, from Finder, and more.

We recommend you [Install agents with the install script](#) on page 147 or [Install agents from the console](#) on page 150 whenever possible. If you want to install macOS agents with PE package management, follow the steps to [Install *nix agents with PE package management](#) on page 151. If you want to install agents from Finder or the macOS command line, follow the steps below.

Install macOS agents from Finder

You can use Finder to install agents on macOS nodes.

1. [Download](#) the appropriate agent tarball.
2. Open the agent package `.dmg` and click the installer `.pkg`.
3. Follow installer dialog prompts. You must provide the primary server's hostname and the agent's `certname`. For macOS agents, the `certname` is derived from the machine's name. For best compatibility, make sure the node's name doesn't include capital letters (for example, `My-Example-Mac` must be `my-example-mac`). If you don't want to change the computer's name, enter the agent `certname` in all lowercase letters when prompted by the installer.
4. Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
5. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Install macOS agents from the command line

You can use the command line to install agents on macOS nodes.

1. [Download](#) the appropriate agent tarball.
2. SSH into the target node as root or sudo.
3. To mount the disk image, run: `sudo hdiutil mount <DMGFILE>`
4. Locate a line ending with `/Volumes/puppet-agent-<VERSION>`. This directory location is the mount point for the virtual volume created from the disk image.
5. Change to the mount point directory, such as with: `cd /Volumes/puppet-agent-<VERSION>`
6. To install the agent package, run: `sudo installer -pkg puppet-agent-installer.pkg -target /`

7. To verify the installation, run: `/opt/puppetlabs/bin/puppet --version`
8. To set the primary server's hostname in the node's `puppet.conf` file, run: `/opt/puppetlabs/bin/puppet config set server <PRIMARY_HOSTNAME>`
Go to [Customize the install script](#) on page 148 for more information about `cpuppet.conf`.
9. To set the agent's `certname` in the node's `puppet.conf` file, run: `/opt/puppetlabs/bin/puppet config set certname <AGENT_CERTNAME>`
For macOS agents, the `certname` is derived from the machine's name. For best compatibility, make sure the node's name doesn't include capital letters (for example, `My-Example-Mac` must be `my-example-mac`). If you don't want to change the computer's name, enter the agent `certname` in all lowercase letters.
10. Run `puppet agent -t` to add the node to the node inventory and generate the CSR.
11. Accept the CSR as explained in [Managing certificate signing requests](#) on page 164.

Install non-root agents

You can configure non-root agents on *nix and Windows nodes. Running agents without root privileges allows teams to perform some, but not all, administrative actions in Puppet Enterprise (PE) that would otherwise require root privileges.

For example, assume a team with root privileges maintains your infrastructure's platform, and a separate team with diminished privileges maintains your infrastructure's applications. If the application team needs to manage their part of the infrastructure independently, they can do this by running Puppet without root privileges.

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 directories where they have write privileges.

By default, PE is installed with root privileges; therefore, a root user must install the agent and configure non-root access to the primary server. The root user also sets up non-root users on the primary server and relevant agent nodes.

Note: In Windows, the administrator is equivalent to the root user. For the sake of simplicity, our documentation might use `root` to refer to either the root user or the administrator.

Non-root user functionality

Non-root users can use a subset of administrative functionality. Non-root agents can't perform any operations requiring root privileges, such as installing system packages.

*nix non-root functionality

Non-root users on *nix agents can enforce these resource types, with some caveats as noted:

- `augeas`
- `cron`: Can only view or set non-root cron jobs
 - If you run a cron job as non-root user and you use the `-u` flag to sets a user with root privileges, the job fails with this error message: `Notice: /Stage[main]/Main/Node[nonrootuser]/Cron[illegal_action]/ensure: created must be privileged to use -u`
- `exec`: Cannot run as another user or group
- `file`: Non-root user must have read/write privileges
- `notify`
- `schedule`
- `service`
- `sshAuthorizedKey`
- `sshKey`

Non-root users on *nix agents can inspect host, mount, and package resource types with the `puppet resource <RESOURCE_TYPE>` command.

Windows non-root functionality

Windows non-root agents are limited in comparison to *nix non-root agents. While you can enforce and inspect some resource types, you are limited to what the agent user has permission to do, which isn't much by default. For example, you can't create a file or directory in C:\Windows unless the agent user has permission to do so.

Non-root users on Windows agents can enforce `exec` and `file` resource types.

Non-root users on Windows agents can use the `puppet resource <RESOURCE_TYPE>` command to inspect these resource types:

- host
- package
- user
- group
- service

Install non-root *nix agents

To configure a *nix agent node to run without root privileges, a root user must install the agent, configure non-root access to the primary server, and set up non-root users on the primary server and relevant agent nodes.

Before you begin

Install the agent on each node you want to operate without root privileges. You can [Install agents with the install script](#) on page 147, [Install agents from the console](#) on page 150, or use one of the other methods to [Install *nix agents](#) on page 151.

Note: Unless specified otherwise, perform these steps as a root user or with `sudo`.

1. Log in to the agent node and run this command to add the non-root user:

```
sudo puppet resource user <UNIQUE_NON-ROOT_USERNAME> ensure=present  
managehome=true
```

Note: Each non-root user must have a unique name.

2. Set the non-root user password. On most *nix systems, you can use `passwd <USERNAME>` to do this.
3. Because the `puppet` service runs as an administrator by default, you must disable it. To stop the `puppet` service run:

```
sudo puppet resource service puppet ensure=stopped enable=false
```

4. Disable the Puppet Execution Protocol (PXP) 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.
 - c) Set the `pxp_enabled` parameter to `false`.
 - d) Click **Add parameter** and commit changes.
5. Switch to the non-root user.

Important: If you use `su - <NON-ROOT USERNAME>` to switch accounts, use the `-` argument (or `-l`, in some Unix variants) to correctly grant full login privileges. Otherwise you might get `permission denied` errors when trying to apply a catalog.

- As the non-root user, run this command to generate a CSR:

```
sudo puppet agent -t --certname "<UNIQUE_NON-ROOT_USERNAME.HOSTNAME>" --server "<PRIMARY_HOSTNAME>"
```

Make sure to format the certname string correctly by combining the non-root user's username and the hostname with a period between.

- On the primary server or in the PE console, approve the CSR.
- On the agent node as the non-root user, run these three commands to set the node's certname, set the primary server's hostname, and run Puppet:

```
sudo puppet config set certname <UNIQUE_NON-ROOT-USERNAME.HOSTNAME> --section agent
sudo puppet config set server <PRIMARY_HOSTNAME> --section agent
sudo puppet agent -t
```

The certname and hostname are defined in the node's `puppet.conf` file.

The configuration specified in the catalog is applied to the agent node.

If you see Facter facts being created in the non-root user's home directory, you have successfully configured a functional non-root agent. To confirm the non-root agent's configuration, verify that:

- The agent can request certificates and apply the catalog from the primary server when a non-root user runs Puppet. As a non-root user, try running `puppet agent -t` to test this.
- The agent service is not running. Run `service puppet status` to check this.
- Non-root users can collect existing facts by running `facter` on the agent.
- Non-root users can define new external facts.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Install non-root Windows agents

To configure a Windows agent node to run without root privileges, a root user must install the agent, configure non-root access to the primary server, and set up non-root users on the primary server and relevant agent nodes.

Before you begin

Install the agent on each node you want to operate without root privileges. You can [Install agents with the install script](#) on page 147, [Install agents from the console](#) on page 150, or use one of the other methods to [Install Windows agents](#) on page 154.

Note: Unless specified otherwise, perform these steps as an administrator.

- Log in to the agent node, open a command prompt as an administrator, and run this command to add the non-root user:

```
puppet resource user <UNIQUE_NON-ADMIN_USERNAME> ensure=present managehome=true password="<PASSWORD>" groups="<EXISTING_GROUP>"
```

Note: Each non-root user must have a unique name.

- Because the `puppet` service runs as an administrator by default, you must disable it. To stop the `puppet` service, open a command prompt as an administrator and run:

```
puppet resource service puppet ensure=stopped enable=false
```

3. Switch to the non-root user and run this command to generate a CSR:

```
puppet agent -t --certname "<UNIQUE_NON-ADMIN_USERNAME.hostname>" --server
"<PRIMARY_HOSTNAME>"
```

Make sure to format the certname string correctly by combining the non-root user's username and the hostname with a period between.

Important: This Puppet run submits a CSR 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.

4. As the non-root user, create a puppet.conf file in the .puppet directory (at %USERPROFILE%/.puppet/). Edit the puppet.conf file and specify the agent certname and the primary server's hostname. For example:

```
[main]
certname = <UNIQUE_NON-ADMIN_USERNAME.hostname>
server = <PRIMARY_HOSTNAME>
```

5. As the non-root user, run `puppet agent -t` to submit a CSR.
6. On the primary server or in the PE console, approve the CSR.

Important: It's possible to sign the root user certificate to allow the non-admin user to also manage the node; however, this is a security concern due to the opportunity for unwanted behavior. For example, if your site.pp has no default node configuration, and a non-admin user runs the agent, unwanted node definitions could be generated with alt hostnames, which is a potential security issue. If you elect to allow non-admin users to also manage nodes, make sure you take precautions such as having clear node definitions, correctly scoping classes, and ensuring root and non-root users never try to manage the same resources.

7. On the agent node as the non-root user, run `puppet agent -t`

The configuration specified in the catalog is applied to the node.

Managing certificate signing requests

When you install a Puppet agent on a node, the agent must submit a certificate signing request (CSR) to the primary server, and you must accept the CSR to add the node to your Puppet Enterprise (PE) inventory. Accepting the CSR 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 PE console.

If you [Install agents from the console](#) on page 150, the agent automatically submits a certificate signing request (CSR) to the primary server. If you use another method, such as [Install agents with the install script](#) on page 147, you might need to run `puppet` to generate the CSR after installing the agent.

You can accept CSRs from the PE console or the command line.

Restriction: For agent nodes that use DNS altnames, you must use the command line to accept the CSR.

If necessary after installing the agent, you can edit the node's certname or other CSR attribute settings in the node's `puppet.conf` and `csr_attributes.yaml` files. You can edit the `puppet.conf` file directly (at /etc/puppetlabs/puppet/puppet.conf) or use the `puppet config set` sub-command. For example, to set the certname for the agent, run /opt/puppetlabs/bin/puppet config set certname agent.example.com. For more information about `puppet.conf` and `csr_attributes.yaml`, go to [Customize the install script](#) on page 148 (This page is about setting these properties with the agent install script, but you can edit these properties after installing the agent).

For information about configuring the certificate authority to automatically sign certain CSRs, refer to [Autosigning certificate requests](#) in the Puppet documentation.

Managing CSRs in the console

In the Puppet Enterprise (PE) console, you can accept or reject CSRs individually or in batches.

Before you begin: You must have the **Console: View** and **Certificate requests: Accept and reject** permissions.

1. In the console, go to **Certificates > Unsigned certificates**.
2. To manage an individual CSR, click **Accept** or **Reject**.
3. To manage all unsigned CSRs at once, click **Accept All** or **Reject All**.

Important: Stay on this page while the CSRs are processed. Nodes are processed in batches, and closing your browser or navigating to another page stops the process after the current batch.

4. To make the node available in the console, manually start a Puppet run or wait for the next scheduled Puppet run.

Related information

[Run Puppet on demand](#) on page 604

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

Managing CSRs on the command line

You can use the command line to view and sign individual CSRs.

Before you begin: You must have the **Certificate requests: Accept and reject** permission.

These instructions use *nix commands. For Windows, run the commands in an administrator command prompt without sudo.

1. To view pending CSRs, run: `sudo puppetserver ca list`
 2. To sign a CSR, run: `sudo puppetserver ca sign --certname <NAME>`
- You can use the Puppet Server CA CLI to sign certificates with altnames or auth extensions by default.
3. To make the node available in the console, run `puppet agent -t` or wait for the next scheduled Puppet run.

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.

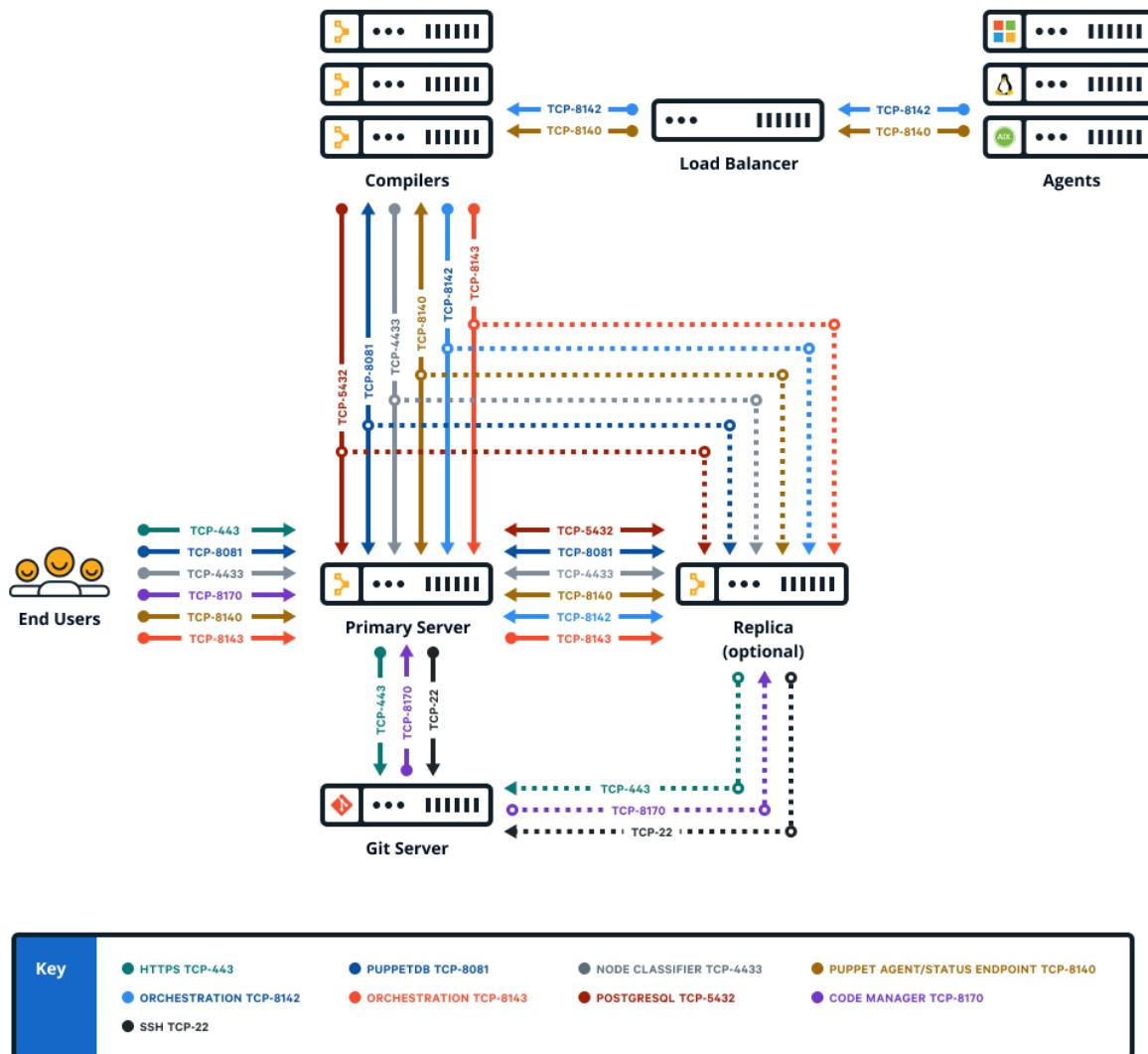
Each compiler increases capacity by 1,500 to 3,000 nodes, until you exhaust the capacity of PuppetDB or the console.

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`. Logback configuration for PCP broker logs is part of the [Orchestration services settings](#) on page 597.

Using load balancers with compilers

When using more than one compiler, a load balancer can help distribute the load between the compilers and provide a level of redundancy.

Specifics on how to configure a load balancer infrastructure falls outside the scope of this document, but examples of how to leverage `haproxy` for this purpose can be found on the [HAproxy module Forge page](#).

Calculating load balancing

For load balancing between the Puppet agent and the Puppet primary server, implement a load balancing algorithm that distributes traffic among compilers based on the number of open connections. Traffic is directed to the compiler with the smallest number of open connections. This strategy is known as “balancing by least connections.”

For load balancing between PCP brokers and PXP agents, a different method is used. PXP agents establish TCP connections to PCP brokers over port 8142. PCP brokers are built on web sockets and require many persistent connections. PCP brokers depend on maintaining connectivity to the Puppet orchestrator, but if the brokers become disconnected from the orchestrator, the brokers can fail at the HTTP level while still accepting TCP connections. Follow these guidelines:

- If you are using HTTP health checks, use a "least connections" algorithm to distribute load evenly.
- If you are not using HTTP health checks, use a round robin or random load balancing algorithm to avoid directing all traffic to an unhealthy PCP broker. You can check connections for possible errors by using the `/status/v1/simple` endpoint.

Using health checks

The Puppet REST API exposes a status endpoint that can be used for load balancer health checks to ensure that unhealthy hosts don't receive agent requests.

To check the health of your hosts, issue HTTP GET requests to the following endpoints:

- For Puppet agent traffic on port 8140, use `https://<hostname>:8140/status/v1/simple/server`. This endpoint checks the health of the `pe-puppetserver` service and returns an HTTP 200 status when the server is fully operational. It does not check the health of other services such as the `broker-service`.
- For `pyp-agent` traffic on port 8142, use `https://<hostname>:8140/status/v1/simple/broker-service`. This endpoint checks the health of the `broker-service` and returns an HTTP 200 status code when the service is fully operational.

If your load balancer doesn't support HTTP health checks, you can use a TCP connection tests on port 8140 to check whether a host is listening. This test is useful for verifying that the port is open, but does not confirm whether the host is running a Puppet service.

Related information

[Firewall configuration for large installations](#) on page 108

These are the port requirements for large installations with compilers.

[GET /status/v1/simple](#) on page 424

Returns a cumulative status reflecting all services the status service knows about.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Load balancing for multi-region installations

If you have load balancers in multiple regions, use a global DNS proximity-based service address.

Tip: For guidance about deploying PE in global, multi-region, or multi-network segment scenarios, see the [Multi-region Reference Architectures](#) article.

When using a centralized Puppet deployment with multiple regional proxies or load balancers, create a global DNS proximity-based service address for Puppet and use that to route agents to the appropriate regional load balancer based on their location. Set the global DNS proximity-based address as the compiler pool address in Hiera.

For example, in your common.yaml file:

```
pe_repo::compile_master_pool_address: "<PUPPET-GLOBAL-SERVICE-ADDRESS>"
```

Some suitable global DNS proximity-based service address implementations include:

- BIG-IP DNS
- Route 53 Geolocation routing in AWS
- TCP Proxy Global Load Balancing in GCP
- Traffic Manager in Azure

Install compilers

Installing a compiler adds the specified node to the **PE Infrastructure Agent** and **PE Compiler** node groups and installs the PuppetDB service on the node.

Before you begin

The node you want to provision as a compiler must have a Puppet agent installed, or you must be able to connect to a non-agent node with SSH.

Ensure that you have a valid admin RBAC token. For instructions, see [Token-based authentication](#) on page 303.

Important: Before you install compilers in multi-region installations, contact Support for guidance . 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.

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.

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
manage_puppet_conf	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.
pcp_broker_list	Hostname for your primary server and replica, if you have one. Hostnames must include port 8142, for example <code>["PRIMARY.EXAMPLE.COM:8142" , "REPLICA.EXAMPLE.COM:8142"]</code> .
primary_uris	Hostname for your primary server and replica, if you have one, for example <code>["PRIMARY.EXAMPLE.COM" , "REPLICA.EXAMPLE.COM"]</code> . This setting assumes port 8140 unless you specify otherwise with <code>host:port</code> .
server_list	

- 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 then run Puppet on the node (run `puppet agent -t`).
3. On your primary server, logged in as root, run the following command to provision a single compiler:

```
puppet infrastructure provision compiler <COMPILER_FQDN>
```

This command accepts a maximum of one compiler FQDN; this command can't provision multiple compilers at once. Additionally, you can specify these optional parameters:

- **dns-alt-names**: Comma-separated list of any alternative names that agents use to connect to the compiler. The installation uses `puppet` by default.

Important: 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 (as indicated by `allow-subject-alt-names: false` in your `ca.conf` file).
- **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.

4. Verify that the contents of the global layer Hiera file on the new compiler, located at `/etc/puppetlabs/puppet/hiera.yaml`, match the contents of the global layer Hiera file on the primary server.
 - If necessary, update `hiera.yaml` on the compiler to match `hiera.yaml` on the primary server.
 - If you use code to manage the contents of `hiera.yaml` on the primary server, ensure that the new compiler is also classified to manage the contents of its own `hiera.yaml` file.

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.

Restriction: This procedure is not intended for installations with load balancers in multiple locations. To configure compilers in multi-region installations, refer to [Load balancing for multi-region installations](#) on page 168.

1. Configure `pe_repo::compile_master_pool_address` to send agent install requests to the load balancer.
 - 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 set the value of the **compile_master_pool_address** parameter to the load balancer hostname. If you are using a single compiler, set the **compile_master_pool_address** value to the compiler's fully qualified domain name (FQDN).
 - c) Click **Add data** and commit changes.
 - d) Run Puppet on the compiler, and then on the primary server.

2. Configure Puppet agents to connect orchestration (PXP) agents to compilers through the load balancer. You can configure these settings in the console or with Hiera.
 - 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 <code>true</code> to ensure that your setting for <code>server_list</code> is configured in the expected location and persists through Puppet runs.
pcp_broker_list	Specify hostnames for your load balancers. This setting configures PXP agents. Hostnames must include port 8142, for example: ["LOADBALANCER1.EXAMPLE.COM:8142", "LOADBALANCER2.EXAMPLE.COM:8142"]
primary_uris	Specify hostnames for your load balancers, for example: ["LOADBALANCER1.EXAMPLE.COM", "LOADBALANCER2.EXAMPLE.COM"]
server_list	This setting configures PXP agents and assumes port 8140 unless you specify otherwise, such as: "LOADBALANCER1.EXAMPLE.COM:<PORT>"
pcp_broker_ws_uris	Specify hostnames for your load balancers, for example: ["LOADBALANCER1.EXAMPLE.COM", "LOADBALANCER2.EXAMPLE.COM"]
server_uris	This setting configures Puppet agents and assumes port 8140 unless you specify otherwise, such as: "LOADBALANCER1.EXAMPLE.COM:<PORT>"

- c) Remove any values set for `pcp_broker_ws_uris`.
- d) Commit changes.
- e) Run Puppet on the primary server, and then run Puppet on all agents or install new agents.

This Puppet run configures both Puppet agents and PXP agents to connect to the load balancer.

Related information

[Firewall configuration for large installations](#) on page 108

These are the port requirements for large installations with compilers.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Convert existing compilers

If you have legacy compilers, you can improve their usability and scalability by adding PuppetDB. In addition to installing the PuppetDB service, converting an existing compiler adds the node to the **PE Compiler** node group and unpins it from the **PE Master** node group.

Before you begin

Open port 5432 from compilers to your primary server or, in extra-large installations, your PE-PostgreSQL node.

Important: Contact Support for guidance before converting compilers in multi-region installations. If your primary server and compilers are connected with high-latency links or congested network segments, you might experience better PuppetDB performance with legacy compilers.

On your primary server logged in as root, run:

```
puppet infrastructure run convert_legacy_compiler
compiler=<COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

Tip: To convert all compilers:

```
puppet infrastructure run convert_legacy_compiler all=true
```

Run `puppet infrastructure tune` on your primary server and adjust tuning for compilers as needed.

Related information

[The `puppet infrastructure tune` command](#) on page 205

The `puppet infrastructure tune` command outputs optimized settings for Puppet Enterprise (PE) services based on recommended guidelines.

Installing client tools

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

The `pe-client-tools` package is included in the PE installation tarball. When you install, the client tools are automatically installed on the same node as the primary server. When you upgrade, client tools are automatically updated on infrastructure nodes and managed nodes, but on *unmanaged* nodes, you must re-install the version of client tools that matches the PE version you upgraded to.

Client tools versions align with PE versions. For example, if you're running PE 2021.7, use the 2021.7 client tools. In some cases, we might issue patch releases ("x.y.z") for PE or the client tools. You don't need to match patch numbers between PE and the client tools. Only the "x.y" numbers need to match.

Note: To see the version of client tools installed on your system, use the command appropriate for your package manager or operating system. For example, on Red Hat: `rpm -q pe-client-tools`.

The package includes client tools for these services:

- Orchestrator — Allow you to control the rollout of changes in your infrastructure, and provides the interface to the orchestration service. Tools include `puppet job` and `puppet task`.
- Puppet access — Authenticates you to the PE RBAC token-based authentication service so that you can use other capabilities and APIs.
- Code Manager — Provides the interface for the Code Manager and file sync services. Tools include `puppet-code`.
- PuppetDB CLI — Enables certain operations with PuppetDB, such as building queries and handling exports.

Because you can safely run these tools remotely, you no longer need to SSH into the primary server to execute commands. Your permissions to see information and to take action are controlled by PE role-based access control. Your activity is logged under your username rather than under root or the `pe-puppet` user.

Related information

[Orchestrator configuration files](#) on page 600

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 303

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 795

Puppet Enterprise (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 workstation, customize configuration for different users, or change the global configuration settings.

Supported PE client tools operating systems

The PE client tools package can be installed on the following platforms.

Operating system	Versions	Arch
Amazon Linux	2, 2023	<ul style="list-style-type: none"> 2: x86_64 2023: amd64
CentOS	6, 7	x86_64
macOS	11, 12, 13, 14	<ul style="list-style-type: none"> 11: x86_64 12: x86_64, M1, M2 13: x86_64, ARM 14: ARM
Microsoft Windows	10, 11	<ul style="list-style-type: none"> 10: x86, x64 11: x64
Microsoft Windows Server	2012, 2012 R2, 2012 R2 Core, 2016, 2016 Core, 2019, 2019 Core, 2022	x64
Oracle Linux	6, 7, 8, 9	x86_64
Red Hat Enterprise Linux	6, 7, 8, 9	x86_64
Scientific Linux	6, 7	x86_64
Solaris	10, 11	<ul style="list-style-type: none"> 10: SPARC, i386 11: SPARC, x86_64
SUSE Linux Enterprise Server	12, 15	x86_64
Ubuntu	18.04, 20.04, 22.04	amd64

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 442

Classification node groups 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, go to the documentation for each service:

Client tool	Documentation
Orchestrator	<ul style="list-style-type: none"> • How Puppet orchestrator works on page 586 • Run Puppet on demand from the CLI on page 611 • Running tasks from the command line on page 622
Puppet access	Token-based authentication on page 303
Puppet code	Triggering Code Manager on the command line on page 795
PuppetDB	PuppetDB CLI

Related information

[Orchestrator configuration files on page 600](#)

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 303](#)

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 795](#)

Puppet Enterprise (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 workstation, customize configuration for different users, or change the global configuration settings.

Uninstalling

Puppet Enterprise (PE) includes a script for uninstalling. You can uninstall infrastructure nodes or uninstall the agent from agent nodes.

Uninstall infrastructure nodes

The `puppet-enterprise-uninstaller` script is installed on the primary server. You must run the uninstaller script on each infrastructure node you want to uninstall.

By default, the uninstaller removes the software, users, logs, cron jobs, and caches. However, the uninstaller **does not remove**:

- Modules
- Manifests

- Certificates
- Databases
- Configuration files
- Home directories of any removed users

If you want to uninstall and reinstall a primary server on the same system, use the appropriate [Uninstaller options](#) on page 178 to remove the otherwise untouched files and databases.

1. On the infrastructure node that you want to uninstall, open the command line as root and navigate to the installer directory at `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`
2. Run the uninstall command: `sudo ./puppet-enterprise-uninstaller`
3. Follow the prompts to complete the uninstallation.
4. Remove the infrastructure node's certificate from the Puppet infrastructure configuration by running the following command on the primary server:

```
puppet node purge <PE_COMPONENT_CERT_NAME>
```

Uninstall agents

You can remove the `puppet-agent` package from nodes that you no longer want Puppet Enterprise (PE) to manage.

Important: Uninstalling the agent doesn't remove the node from your inventory or free up the node's license to use on another node. To remove a node, you must purge the node.

Related information

[Adding and removing agent nodes](#) on page 432

You can add nodes you want to manage with Puppet Enterprise (PE) and remove nodes you no longer need.

[Installing agents](#) on page 145

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

Uninstall *nix agents

The *nix agent package includes an uninstall script you can use to remove the agent from the node.

1. On the agent node, run the uninstall script: `/opt/puppetlabs/bin/puppet-enterprise-uninstaller`
2. Follow the prompts to complete the uninstallation.
3. Optional: If you want to reinstall the agent on the node later, make sure you remove the node's agent certificate from the primary server. To do this, on the primary server run: `puppetserver ca clean --certname <AGENT_CERT_NAME>`

If you want to remove the node from your inventory and use the node's license on another node, you must purge the node, as explained in [Remove agent nodes](#) on page 433.

Uninstall Windows agents

To uninstall the agent from a Windows node, use the Windows **Add or Remove Programs** interface or the command line.

Uninstalling the agent from a Windows node 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 the Puppet agent from the system, you must also manually delete the data directory.

1. Uninstall the agent:

- Use the Windows **Add or Remove Programs** interface.
- Use the command line if you have the original `.msi` file or know the installed MSI's product code. For example: `msiexec /qn /norestart /x [puppet.msi]<PRODUCT_CODE>`

2. Optional: If you want to reinstall the agent on the node later, make sure you remove the node's agent certificate from the primary server. To do this, on the primary server run: `puppetserver ca clean --certname <AGENT_CERT_NAME>`

If you want to remove the node from your inventory and use the node's license on another node, you must purge the node, as explained in [Remove agent nodes](#) on page 433.

Uninstall macOS agents

Use the command line to remove 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 want to reinstall the agent on the node later, make sure you remove the node's agent certificate from the primary server. To do this, on the primary server run: `puppetserver ca clean --certname <AGENT_CERT_NAME>`

If you want to remove the node from your inventory and use the node's license on another node, you must purge the node, as explained in [Remove agent nodes](#) on page 433.

Uninstaller options

You can use these command line options to change the uninstaller's behavior.

- `-p`: Purge additional files. In addition to the software, users, logs, cron jobs, and caches, 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`: Remove any databases created during installation.
- `-h`: Display a help message.
- `-n`: Run in noop mode and show commands that would have run during uninstallation without actually running them.
- `-y`: Don't ask to confirm uninstallation. Assuming the answer is yes.

To remove every trace of PE from a system, which is required if you want to reinstall PE on the same system, run this command:

```
sudo ./puppet-enterprise-uninstaller -d -p
```

For Windows systems, open an administrator command prompt and run the command without sudo.

Related information

[Using example commands](#) on page 28

These guidelines can help you understand and customize the example commands you'll find in the Puppet Enterprise (PE) docs.

Upgrading

To upgrade your Puppet Enterprise deployment, you must upgrade both the infrastructure components and agents.



CAUTION:

Major primary server OS upgrades (such as Ubuntu 18.04 to 20.04) require [Back up and restore PE](#) on page 846.

Major agent OS upgrades require reinstalling the `puppet-agent` package (as explained in [Installing agents](#) on page 145) and reinstalling any Ruby plugins/gems that were added at `/opt/puppetlabs/puppet/bin/gem`.

- [Upgrade paths](#) on page 179

These are the valid upgrade paths for PE.

- [Upgrade cautions](#) on page 180

These are the major changes to PE since the previous long-term support release, 2019.8. Review these recommendations and plan accordingly before upgrading to this version.

- [Test modules before upgrade](#) on page 182

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

- [Upgrading Puppet Enterprise](#) on page 182

Upgrade your PE installation as new versions become available.

- [Upgrading agents](#) on page 196

Upgrade your agents as new versions of Puppet Enterprise (PE) become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can use a script to upgrade individual nodes.

- [Migrate PE](#) on page 201

As an alternative to upgrading, you can *migrate* your PE installation. Migrating results in little or no downtime, but it requires additional system resources because you must configure a new primary server.

Upgrade paths

These are the valid upgrade paths for PE.

If you're on version...	Upgrade to...	Notes
2021.7.10	You're up to date!	
2019.8.z or any 2021.y	2021.7.z	Before upgrading to 2021.7.z, you might want to read about What's new since PE 2019.8 on page 55.
2019.y	2019.8.12	

If you're on version...	Upgrade to...	Notes
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		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.
2018.1.0 or 2018.1.1	2018.1.z	

Related information

[Component versions in recent PE releases](#) on page 14

These tables show which components are in recent Puppet Enterprise (PE) long-term supported (LTS) releases and the prior LTS releases (2019.8.z). Component version tables for earlier releases are available in the [Documentation for other PE versions](#) on page 31. Component version tables for newer releases are available in the [latest documentation](#).

[Server and agent compatibility](#) on page 16

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

Upgrade cautions

These are the major changes to PE since the previous long-term support release, 2019.8. Review these recommendations and plan accordingly before upgrading to this version.

FIPS-enabled PE 2021.7 and later can't use the default system cert store

PE 2021.7 and later FIPS builds can't use the default system cert store, which is used automatically with some reporting services. This setting is configured by the `report_include_system_store` Puppet parameter that ships with PE.

Removing the `puppetcacerts` file (located at `/opt/puppetlabs/puppet/ssl/puppetcacerts`) can allow a report processor that eagerly loads the system store to continue with a warning that the file is missing.

If HTTP clients require external certs, we recommend using a custom cert store containing only the necessary certs. You can create this cert store by concatenating existing pem files and configuring the `ssl_trust_store` Puppet parameter to point to the new cert store.

Update `puppet_agent` module to support AIX

If you use the `puppet_agent` module and have the agent installed on any AIX nodes, then before you upgrade to PE 2021.7.z, you must ensure that you are using `puppet_agent` module version 4.18.0 or later. This ensures that the `puppet_agent` module identifies the correct directory for AIX resources and your AIX agents function as expected.

Logback upgrade in PE 2021.7.7 and later

In PE 2021.7.7, logback is upgraded to version 1.3.14. Using a Java argument, the `logappender` variable is now set by default to `F1` for all projects. If you customize this setting, to avoid disruptions in logging, ensure that all `logappender` variable references are correctly defined. Using invalid appender references or omitting to use a reference will cause logback version 1.3.14 to stop logging.

PostgreSQL 14 upgrade in PE 2021.6

PE 2021.6.0 upgrades `pe-postgresql` from version 11 to version 14. This upgrade involves a datastore migration that requires extra disk space (110% of the current PostgreSQL 11 datastore) and extra time to upgrade (roughly two

to four minutes of additional time per 10GB of datastore size). The installer issues a warning and cancels the upgrade if there is insufficient space.

To check your PostgreSQL installation size and the size and number of bytes available for the partition, run `facterc -p pe_postgresql_info` on the node that runs the `pe-postgresql` service (this is either your primary server or a separate node that manages your PostgreSQL database).

To speed the migration and optimize queries, clean up the PE-PostgreSQL database prior to upgrading by applying the `pe_databases` module to nodes running the `pe-postgresql` service. This module is installed with PE versions 2021.3 and later, but you must enable it by setting the `puppet_enterprise::enable_database_maintenance` parameter to `true`. For best results, apply the module at least one week prior to upgrade to allow the module's maintenance schedule enough time to clean all databases.

Optionally, after upgrading, you can remove packages and directories from older PostgreSQL versions by running `puppet infrastructure run remove_old_postgresql_versions`. If applicable, the installer prompts you to complete the cleanup.



CAUTION: Don't use `CONCURRENTLY` with PostgreSQL 14.0 through 14.3. If you do, make sure to `REINDEX` without using `CONCURRENTLY`.

Restriction: If you use a PostgreSQL instance not managed by PE, you must separately upgrade the instance to PostgreSQL 14. For more information, see [Upgrade an unmanaged PostgreSQL installation](#) on page 187.

Platforms removed in 2021.0 and later

Several agent platforms that were previously deprecated have been removed in PE 2021.0 and later.

Before upgrading to PE 2021.7.5, 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.

Platforms removed in 2021.0

Removed agent platforms

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

Platforms removed in 2021.7.5

Removed agent platforms

- CentOS 7 aarch64
- macOS 10.15
- Oracle Linux 7 aarch64
- Red Hat 7 aarch64
- Scientific Linux 7 aarch64

Puppet upgrade in 2021.0

PE 2021.0 includes a new major version of Puppet, with several changes that might impact your upgrade. For details, see [Upgrading from Puppet 6 to Puppet 7](#).

Test modules before upgrade

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

Before you begin

If you are already using PDK, your modules should pass validation and unit tests with your currently installed version of PDK.

Update PDK with each new release to ensure compatibility with new versions of PE.

1. Download and install PDK. If you already have PDK installed, this updates PDK to its latest version. For detailed instructions and download links, see the [installing](#) instructions.
2. If you have not previously used PDK with your modules, convert them to a PDK compatible format. This makes changes to your module to enable validation and unit testing with PDK. For important usage details, see the [converting modules](#) documentation.

For example, from within the module directory, run:

```
pdk convert
```

3. If your modules are already compatible with PDK, update them to the latest module template. If you converted modules in step 2, you do not need to update the template. To learn more about updating, see the [updating module templates](#) documentation.

For example, from within the module directory, run:

```
pdk update
```

4. Validate and run unit tests for each module, specifying the version of PE you are upgrading to. When specifying a PE version, be sure to specify at least the year and the release number, such as 2018.1. For information about module validations and testing, see the [validating and testing modules](#) documentation.

For example, from within the module directory, run:

```
pdk validate
pdk test unit
```

The `pdk test unit` command verifies that testing dependencies and directories are present and runs the unit tests that you write. It does not create unit tests for your module.

5. If your module fails validation or unit tests, make any necessary changes to your code.

After you've verified that your modules work with the new PE version, you can continue with your upgrade.

Upgrading Puppet Enterprise

Upgrade your PE installation as new versions become available.

- [Upgrade PE using the installer tarball](#) on page 183

Upgrade PE infrastructure components to get the latest features and fixes. Follow the upgrade instructions for your installation type to ensure you upgrade components in the correct order. Coordinate upgrades to ensure all infrastructure nodes are upgraded in a timely manner, because agent runs and replication fail on infrastructure nodes running a different agent version than the primary server.

- [Upgrade PE using PIM](#) on page 191

Puppet Installation Manager (PIM) supports the upgrading of Puppet Enterprise (PE) for all supported installation architectures. For an interactive experience, choose the guided upgrade process and follow the steps in your terminal. Alternatively, if you do not require guidance, you can run your upgrade from the PIM command line by passing a JSON file containing your installation parameters.

Upgrade PE using the installer tarball

Upgrade PE infrastructure components to get the latest features and fixes. Follow the upgrade instructions for your installation type to ensure you upgrade components in the correct order. Coordinate upgrades to ensure all infrastructure nodes are upgraded in a timely manner, because agent runs and replication fail on infrastructure nodes running a different agent version than the primary server.

Before you begin

Review the [upgrade cautions](#) for major changes to architecture and infrastructure components which might affect your upgrade.

Configure non-production environment for infrastructure nodes

If your infrastructure nodes are in an environment other than production, you must manually configure PE to use your chosen environment before you upgrade.

Important: You only need to follow these steps if your infrastructure nodes are in an environment that is not production.

In `pe.conf`, set both of these parameters to the environment your infrastructure nodes are in:

- `pe_install::install::classification::pe_node_group_environment`
- `puppet_enterprise::master::recover_configuration::pe_environment`

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.

In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.

1. [Download](#) the tarball for your operating system and architecture. Optionally, you can [Verify the installation package](#) on page 125.
2. Run `tar -xf <TARBALL>` to unpack the installation tarball.
You need about 1 GB of space to untar the installer.
3. From the installer directory on your primary server, run `sudo ./puppet-enterprise-installer` to start the installer, and then follow the CLI instructions to complete your server upgrade.

To specify a different `pe.conf` file than the existing file, use the `-c` flag, such as:

```
sudo ./puppet-enterprise-installer -c <FULL_PATH_TO_pe.conf>
```

This flag tells the installer to backup the previous `pe.conf` file to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and create a new `pe.conf` file at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

4. Upgrade these additional PE infrastructure components:

- Agents
- PE client tools: On unmanaged nodes, you must re-install the client tools version that matches the PE version you upgraded to. On managed nodes and infrastructure nodes, client tools are automatically updated when you upgrade PE.

5. In disaster recovery installations, upgrade your replica.

The replica is temporarily unavailable to serve as backup during this step, so time upgrading your replica to minimize risk.

- a) On your primary server logged in as root, run:

```
sudo puppet infrastructure upgrade replica <REPLICA_FQDN>
```

If you want to specify an authentication token other than the default, run:

```
sudo puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file
<PATH_TO_TOKEN>
```

- b) After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run:

```
sudo /opt/puppetlabs/bin/puppet-infra status
```

- c) If your replica reports errors, reinitialize the replica. On your replica, run:

```
sudo /opt/puppetlabs/bin/puppet-infra reinitialize replica -y
```

6. Optional: Remove old PE packages from all infrastructure nodes. On your primary server, run: `puppet infrastructure run remove_old_pe_packages`

All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>` to the command.

Related information

[Back up your infrastructure](#) on page 847

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

[Generate a token using puppet-access](#) on page 305

Use the `puppet-access` command 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.

In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.

1. [Download](#) the tarball for your operating system and architecture. Optionally, you can [Verify the installation package](#) on page 125.
2. Run `tar -xf <TARBALL>` to unpack the installation tarball.
You need about 1 GB of space to untar the installer.

- From the installer directory on your primary server, run `sudo ./puppet-enterprise-installer` to start the installer, and then follow the CLI instructions to complete your server upgrade.

To specify a different `pe.conf` file than the existing file, use the `-c` flag, such as:

```
sudo ./puppet-enterprise-installer -c <FULL_PATH_TO_pe.conf>
```

This flag tells the installer to backup the previous `pe.conf` file to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and create a new `pe.conf` file at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

- To upgrade compilers, log in to your primary server as root and run one of these commands:

- To upgrade specific compilers, run:

```
sudo puppet infrastructure upgrade compiler
<COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

- To upgrade all compilers simultaneously, run:

```
sudo puppet infrastructure upgrade compiler --all
```

- To specify an authentication token location other than the default location, include `--token-file <PATH_TO_TOKEN>` in the command, such as:

```
sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file
<PATH_TO_TOKEN>
```

- Upgrade these additional PE infrastructure components:

- Agents
- PE client tools: On unmanaged nodes, you must re-install the client tools version that matches the PE version you upgraded to. On managed nodes and infrastructure nodes, client tools are automatically updated when you upgrade PE.

- 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.

- On your primary server logged in as root, run:

```
sudo puppet infrastructure upgrade replica <REPLICA_FQDN>
```

If you want to specify an authentication token other than the default, run:

```
sudo puppet infrastructure upgrade replica <REPLICA_FQDN> --token-file
<PATH_TO_TOKEN>
```

- After the replica upgrade successfully completes, verify that primary and replica services are operational. On your primary server, run:

```
sudo /opt/puppetlabs/bin/puppet-infra status
```

- If your replica reports errors, reinitialize the replica. On your replica, run:

```
sudo /opt/puppetlabs/bin/puppet-infra reinitialize replica -y
```

- Optional: Remove old PE packages from all infrastructure nodes. On your primary server, run: `puppet infrastructure run remove_old_pe_packages`

All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>` to the command.

Optionally convert legacy compilers to the new style compiler running the PuppetDB service.

Related information

[Back up your infrastructure](#) on page 847

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

[Generate a token using puppet-access](#) on page 305

Use the `puppet-access` command 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

You can use the [PEADM module](#) to upgrade extra-large installations. Contact your technical account manager for additional support.

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.

In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.

1. [Download](#) the tarball for your operating system and architecture. Optionally, you can [Verify the installation package](#) on page 125.
2. Run `tar -xf <TARBALL>` to unpack the installation tarball.
You need about 1 GB of space to untar the installer.
3. Upgrade your PostgreSQL node.
 - a) Ensure that the `user_data.conf` file on your PostgreSQL node is up to date by running `puppet infrastructure recover_configuration` on your primary server, and then copying `/etc/puppetlabs/enterprise/conf.d` to the PostgreSQL node.
 - b) Copy the installation tarball to the PostgreSQL node, and from the installer directory, run the installer:
`sudo ./puppet-enterprise-installer`
4. From the installer directory on your primary server, run `sudo ./puppet-enterprise-installer` to start the installer, and then follow the CLI instructions to complete your server upgrade.

To specify a different `pe.conf` file than the existing file, use the `-c` flag, such as:

```
sudo ./puppet-enterprise-installer -c <FULL_PATH_TO_pe.conf>
```

This flag tells the installer to backup the previous `pe.conf` file to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and create a new `pe.conf` file at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

- To upgrade compilers, log in to your primary server as root and run one of these commands:

- To upgrade specific compilers, run:

```
sudo puppet infrastructure upgrade compiler
  <COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

- To upgrade all compilers simultaneously, run:

```
sudo puppet infrastructure upgrade compiler --all
```

- To specify an authentication token location other than the default location, include `--token-file <PATH_TO_TOKEN>` in the command, such as:

```
sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file
  <PATH_TO_TOKEN>
```

- Upgrade these additional PE infrastructure components:

- Agents
- PE client tools: On unmanaged nodes, you must re-install the client tools version that matches the PE version you upgraded to. On managed nodes and infrastructure nodes, client tools are automatically updated when you upgrade PE.

- Optional: Remove old PE packages from all infrastructure nodes. On your primary server, run: `puppet infrastructure run remove_old_pe_packages`

All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>` to the command.

Optionally [convert legacy compilers](#) to the new style compiler running the PuppetDB service.

Related information

[Back up your infrastructure](#) on page 847

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

[Generate a token using puppet-access](#) on page 305

Use the `puppet-access` command to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Upgrade an unmanaged PostgreSQL installation

To upgrade a Puppet Enterprise (PE) installation that relies on an unmanaged PostgreSQL database, you must upgrade PostgreSQL to version 14 and ensure that your PostgreSQL database roles, privileges, extensions, and configuration are up to date with the most recent version of PE. Then, proceed with the PE upgrade.

Before you begin

Back up your PE installation, and, if desired, back up your PostgreSQL databases.

Ensure that you have a valid role-based access control (RBAC) admin token to upgrade compilers (for example, [Generate a token using puppet-access](#) on page 305).

In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.

- If you are upgrading from a 2019.8 version of PE, ensure that you have upgraded to the latest 2019.8.z version. As part of the upgrade to 2019.8.z, you must have followed the unmanaged PostgreSQL upgrade instructions for that 2019.8 version of PE before you begin the upgrade to 2021.

2. If you are upgrading from a 2021 version of PE that is earlier than 2021.2, add the PuppetDB `read_user`. To add the `read_user`, run the commands that correspond to your authentication method to generate the role and authentication, and then run the grant commands:

- **Password authentication:** Run the following command and ensure that you specify a password:

```
CREATE ROLE "pe-puppetdb-read"
LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER CONNECTION LIMIT -1 PASSWORD
'<PUPPETDB_USER_PASSWORD>' ;
```

- **Certificate authentication:** Create the database user by adding the following lines to the `pg_hba.conf` file:

```
hostssl pe-puppetdb pe-puppetdb-read 0.0.0.0/0
cert map=pe-puppetdb-pe-puppetdb-read-map clientcert=verify-full
hostssl pe-puppetdb pe-puppetdb-read ::/0 cert map=pe-puppetdb-pe-
puppetdb-read-map clientcert=verify-full
```

Map the database user role by adding the following line to the `pg_ident.conf` file:

```
pe-puppetdb-pe-puppetdb-read-map <PRIMARY_CERTNAME> pe-puppetdb-read
```

Create the role by running the following command. Use the `psql` tool:

```
CREATE ROLE "pe-puppetdb-read" LOGIN NOCREATEROLE NOCREATEDB
NOSUPERUSER CONNECTION LIMIT -1;
```

After you complete either of the previous steps, finish granting permissions to the new role by running the following commands:

```
GRANT CONNECT ON DATABASE "pe-puppetdb" TO "pe-puppetdb-read";
ALTER DEFAULT PRIVILEGES FOR USER "pe-puppetdb" IN SCHEMA "public" GRANT
SELECT ON TABLES TO "pe-puppetdb-read";
ALTER DEFAULT PRIVILEGES FOR USER "pe-puppetdb" IN SCHEMA "public" GRANT
EXECUTE ON FUNCTIONS TO "pe-puppetdb-read";
ALTER DEFAULT PRIVILEGES FOR USER "pe-puppetdb" IN SCHEMA "public" GRANT
USAGE ON SEQUENCES TO "pe-puppetdb-read";
GRANT SELECT ON ALL TABLES IN SCHEMA "public" TO "pe-puppetdb-read";
GRANT USAGE ON ALL SEQUENCES IN SCHEMA "public" TO "pe-puppetdb-read";
```

3. If you are upgrading from a 2021 version of PE that is earlier than 2021.4, add the `pgcrypto` extension to the Puppet orchestrator database. Run the following commands by using the `psql` tool:

```
\c "pe-orchestrator"
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
```

4. If you are upgrading from any PE version earlier than 2021.6, upgrade PostgreSQL to version 14:

- a) On your PostgreSQL node, install these packages:

- postgresql14
- postgresql14-libs
- postgresql14-contrib
- postgresql14-server

- b) Stop the PostgreSQL 11 service:

```
systemctl stop postgresql-11.service
```

- c) As the PostgreSQL user, create a database:

```
su - postgres /usrpgsql-14/bin/pg_ctl -D /var/lib/pgsql/14/data/ initdb
```

- d) If you use certificate authentication, note the location of your certificate, private key, and certificate authority files. You might have to move these files to the new installation directory, and you might have to update the `postgresql.conf` file.
- e) Copy your `postgresql.conf` file from the `/var/lib/pgsql/11/data/` directory to the `/var/lib/pgsql/14/data/` directory and update the `data_directory` setting. If you moved the certificate, private key, and certificate authority files, update the `ssl_cert_file`, `ssl_key_file`, and `ssl_ca_file` settings.
- f) Migrate databases to PostgreSQL 14:

```
/usrpgsql-14/bin/pg_upgrade -b /usrpgsql-11/bin -B /usrpgsql-14/bin/ -d /var/lib/pgsql/11/data -D /var/lib/pgsql/14/data/
```

Tip: This command uses `pg_upgrade` to upgrade and migrate your unmanaged PostgreSQL instance from version 11 to 14. If you don't want to use this command, you can back up your PostgreSQL 11 databases, manually wipe the PostgreSQL 11 installation, ensure that PostgreSQL 14 is installed, and then restore the databases to the PostgreSQL 14 installation. Or, you can back up the PostgreSQL 11 databases, set up a new node with PostgreSQL 14, restore databases to the new node, and then reconfigure PE to point to the new `database_host` node.

- g) Copy your `pg_hba.conf` and `pg_ident.conf` files from the `/var/lib/pgsql/11/data/` directory to the `/var/lib/pgsql/14/data/` directory.
- h) In the `pg_hba.conf` file, change all instances of `clientcert=1` to `clientcert=verify-full`.

5. Start the PostgreSQL 14 service:

```
systemctl start postgresql-14
```

6. Optional: As the PostgreSQL user, clean up the database:

```
su - postgres /usrpgsql-14/bin/vacuumdb --analyze --all
```

- If you are upgrading from a version of PE earlier than 2021.7.2, ensure that the `pe-puppetdb-read` role is granted connect by the `pe-puppetdb-migrator` role.

This step is necessary so that the `pe-puppetdb-migrator` can revoke query connections during migrations. Additionally, the chain of `pe-puppetdb-read` role granted to `pe-puppetdb` (and `pe-puppetdb` granted to `pe-puppetdb-migrator`, already present in the database schema) makes it possible for the `pe-puppetdb` and `pe-puppetdb-migrator` roles to kill queries during migration and garbage collection.

In the PostgreSQL cluster, run the following commands:

```
REVOKE CONNECT ON DATABASE "pe-puppetdb" FROM "pe-puppetdb-read";
SET ROLE "pe-puppetdb-migrator";
GRANT CONNECT ON DATABASE "pe-puppetdb" TO "pe-puppetdb-read";
SET ROLE "postgres";
GRANT "pe-puppetdb-read" TO "pe-puppetdb"
```

- [Download](#) the tarball for your operating system and architecture. Optionally, you can [Verify the installation package](#) on page 125.

- Run `tar -xf <TARBALL>` to unpack the installation tarball.

You need about 1 GB of space to untar the installer.

- From the installer directory on your primary server, run `sudo ./puppet-enterprise-installer -s` to start the installer, and then follow the CLI instructions to complete the server upgrade.

- The `-s` flag tells the installer to skip database checks.
- To specify a different `pe.conf` file than the existing file, use the `-c` flag as shown here:

```
sudo ./puppet-enterprise-installer -c <FULL_PATH_TO_pe.conf>
```

This flag tells the installer to back up the previous `pe.conf` file to `/etc/puppetlabs/enterprise/conf.d/<TIMESTAMP>.conf` and create a new `pe.conf` file at `/etc/puppetlabs/enterprise/conf.d/pe.conf`.

- To upgrade compilers, log in to your primary server as root and run one of these commands:

- To upgrade specific compilers, run:

```
sudo puppet infrastructure upgrade compiler
  <COMPILER_FQDN-1>,<COMPILER_FQDN-2>
```

- To upgrade all compilers simultaneously, run:

```
sudo puppet infrastructure upgrade compiler --all
```

- To specify an authentication token location other than the default location, include `--token-file <PATH_TO_TOKEN>` in the command, such as:

```
sudo puppet infrastructure upgrade compiler <COMPILER_FQDN> --token-file
  <PATH_TO_TOKEN>
```

- Upgrade these additional PE infrastructure components:

- Agents
- PE client tools: On unmanaged nodes, you must re-install the client tools version that matches the PE version you upgraded to. On managed nodes and infrastructure nodes, client tools are automatically updated when you upgrade PE.

- Optional: Remove old PE packages from all infrastructure nodes. On your primary server, run: `puppet infrastructure run remove_old_pe_packages`

All packages older than the current version are removed by default. To remove specific versions, append `pe_version=<VERSION_NUMBER>` to the command.

Optionally, [convert existing compilers](#) to the new style compiler running the PuppetDB service.



CAUTION: Don't use CONCURRENTLY with PostgreSQL 14.0 through 14.3. If you do, ensure that you REINDEX without using CONCURRENTLY.

Upgrade PE using PIM

Puppet Installation Manager (PIM) supports the upgrading of Puppet Enterprise (PE) for all supported installation architectures. For an interactive experience, choose the guided upgrade process and follow the steps in your terminal. Alternatively, if you do not require guidance, you can run your upgrade from the PIM command line by passing a JSON file containing your installation parameters.

Regardless of the upgrade process you choose, you can use PIM on a jump host to upgrade PE infrastructure components on remote infrastructure nodes. Alternatively, you can upgrade PE by using PIM on your primary server. In this scenario, if you have additional infrastructure nodes that host PE components, your primary server can serve as a jump host.

PIM uses the Puppet Enterprise Administration Module (PEADM), which depends on Puppet Bolt, a tool for automating Puppet infrastructure maintenance tasks. When you use PIM to upgrade, PIM checks whether Bolt is installed and whether your current installation is configured by PEADM. If necessary, PIM provides the option to automatically install Bolt and convert your installation to a PEADM-compatible configuration, so you can proceed with upgrading.

Upgrade PE using the guided process

For an interactive experience, use the guided upgrade process. PIM fetches information about your current installation configuration and configures your upgrade accordingly.

Before you begin

- Carefully review [Upgrade cautions](#) on page 180.
- Check that your modules work with the new PE version. See [Test modules before upgrade](#) on page 182.
- In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.
- Back up your installation.
- Ensure that you have the required access to the PE infrastructure nodes in your installation:
 - To upgrade PE by using PIM on your primary server, you must log in to your primary server as the root user.
 - To upgrade PE components on remote infrastructure nodes, the machine running PIM must have root SSH access to those nodes.

To upgrade PE by using the PIM guided process:

1. Download the latest version of PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Start the guided upgrade process.

In your terminal, navigate to the `pim` directory and run the following command:

```
./pim wizard
```

3. Follow the guided steps in your terminal to complete the upgrade.

If you require additional guidance during the upgrade process, you can view help content by pressing **Ctrl+H**.

Important: PIM uses the Puppet Enterprise Administration Module (PEADM), which depends on Puppet Bolt, a tool for automating Puppet infrastructure maintenance tasks. PIM checks whether Bolt is installed and whether your current installation is configured by PEADM. If necessary, PIM provides the option to automatically install Bolt and convert your installation to a PEADM-compatible configuration, so you can proceed with upgrading.

Upgrade PE with your defined parameters

If you do not require guidance to upgrade PE, you can specify your upgrade parameters in a JSON file. Then use PIM to start the upgrade by running a single command.

Before you begin

- Carefully review [Upgrade cautions](#) on page 180.
- Check that your modules work with the new PE version. See [Test modules before upgrade](#) on page 182.
- In Hiera, `pe.conf`, or the console (in the **PE Master** node group), remove any `agent_version` parameters you set in the `pe_repo` class that matches your infrastructure nodes. This ensures the upgrade isn't blocked by attempting to download non-default agent versions for your infrastructure OS and architecture.
- Back up your installation.
- Ensure that you have the required access to the PE infrastructure nodes in your installation:
 - To upgrade PE by using PIM on your primary server, you must log in to your primary server as the root user.
 - To upgrade PE components on remote infrastructure nodes, the machine running PIM must have root SSH access to those nodes. You can configure SSH, or use the `-b` flag to pass the SSH key or SSH credentials when you run the upgrade command.
- Ensure that your installation is compatible with PIM. You can proceed directly with upgrading if you installed PE using PIM or the Puppet Enterprise Administration Module (PEADM), or if you previously converted your installation for compatibility. If your installation is not compatible with PIM, you can use the PIM CLI to convert your installation and then proceed with upgrading. See [Converting your installation](#) on page 194.

To upgrade PE from the PIM command line:

1. Download the latest version of PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Create a JSON file specifying the relevant parameters for your PE installation and the version you want to upgrade to.

If you have a saved JSON file that you previously used to install or upgrade PE, you can edit that file as required and use it for your upgrade.

For examples illustrating the JSON properties required for different PE architectures, see [Creating an upgrade parameters file](#) on page 193.

3. Start the upgrade.

In your terminal, navigate to the `pim` directory and run one of the following commands, replacing `parameters.json` with the actual file name (including the file path, if necessary):

- To run the upgrade without debugging and without configuring SSH, use a command like the following example:

```
./pim upgrade parameters.json
```

- To enable debug logging, add `-d` or `--debug`. For example:

```
./pim upgrade parameters.json --debug
```

- To pass an SSH key or SSH credentials for accessing remote nodes, use the `-b` flag as shown in the following examples:

```
./pim upgrade -b user=root -b private-key=~/.ssh/ssh_key parameters.json
```

```
./pim upgrade -b user=root -b password=ssh_password parameters.json
```

- Follow the CLI prompts to complete the upgrade process.

Important: PIM uses the Puppet Enterprise Administration Module (PEADM), which depends on Puppet Bolt, a tool for automating Puppet infrastructure maintenance tasks. When you run the `./pim upgrade` command, PIM checks whether Bolt is installed. If necessary, PIM provides the option to automatically install Bolt so you can proceed with upgrading.

Creating an upgrade parameters file

To upgrade PE from the Puppet Installation Manager (PIM) command line, you must use a JSON file containing your installation parameters and pass that file with the `./pim upgrade` command. The JSON file defines your installation architecture, including the option for disaster recovery, and specifies the PE version you want to upgrade to.

Important: Creating a JSON file containing upgrade parameters is not required if you use the guided upgrade process. With the guided process, PIM automatically fetches information about your current installation configuration and configures your upgrade accordingly.

Upgrade configuration examples

The following examples illustrate how to structure the JSON file for different PE configurations.

Upgrade parameters for an extra-large architecture with disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "replica_host": "pe-xl-core-2.lab1.puppet.vm",
  "replica_postgresql_host": "pe-xl-core-3.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
  "version": "2023.6.0"
}
```

Upgrade parameters for an extra-large architecture without disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
  "version": "2023.6.0"
}
```

Upgrade parameters for a large architecture with disaster recovery

```
{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
  "replica_host": "pe-l-core-2.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
  ],
  "version": "2023.6.0"
}
```

Upgrade parameters for a large architecture without disaster recovery

```
{
  "primary_host": "pe-l-core-0.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
  ],
  "version": "2023.6.0"
}
```

Upgrade parameters for a standard architecture with disaster recovery

```
{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "replica_host": "pe-core-2.lab1.puppet.vm",
  "version": "2023.6.0"
}
```

Upgrade parameters for a standard architecture without disaster recovery

```
{
  "primary_host": "pe-core-0.lab1.puppet.vm",
  "version": "2023.6.0"
}
```

Converting your installation

Puppet Installation Manager (PIM) uses the Puppet Enterprise Administration Module (PEADM), which is a set of Bolt plans for deploying and managing Puppet Enterprise (PE) infrastructure. To use PIM to upgrade, you might be required to convert your installation to a PEADM-compatible configuration.

Converting your installation does not add or remove any PE components and does not alter your installation architecture, but it does implement some required configuration changes, including certificate extensions for your infrastructure nodes and additional node groups for any compilers and database servers included in your installation.

When you use the guided upgrade process, if conversion is required, PIM notifies you of the requirement and runs the conversion automatically when you confirm that you want to proceed.

When using the CLI, if the configuration of your existing installation is not compatible with PIM, you can run a conversion first, and then run the upgrade with your defined parameters.

Convert your installation

Before you begin

- Back up your installation.
- Ensure that you have the required access to the PE infrastructure nodes in your installation.
 - To convert PE by using PIM on your primary server, you must log in to your primary server as the root user.
 - To convert a PE installation that includes remote infrastructure nodes, the machine running PIM must have root SSH access to those nodes. You can configure SSH, or use the `-b` flag to pass the SSH key or SSH credentials when you run the convert command.

To convert your installation by using PIM:

1. Download the latest version of PIM.

Go to the [Puppet Installation Manager download page](#) and download the binary for your operating system.

2. Create a JSON file containing the relevant parameters for your PE installation.

For examples illustrating the JSON properties required for different PE architectures, see [Creating a conversion parameters file](#) on page 195.

3. Start the conversion.

In your terminal, navigate to the `pim` directory and run one of the following commands, replacing `parameters.json` with the actual file name (including the file path, if necessary):

- To run the conversion without debugging and without configuring SSH, use a command like the following example:

```
./pim convert parameters.json
```

- To enable debug logging, add `-d` or `--debug`. For example:

```
./pim convert parameters.json --debug
```

- To pass an SSH key or SSH credentials for accessing remote nodes, use the `-b` flag as shown in the following examples:

```
./pim convert -b user=root -b private-key=~/ssh/ssh_key parameters.json
```

```
./pim convert -b user=root -b password=ssh_password parameters.json
```

Creating a conversion parameters file

To convert your installation using the Puppet Installation Manager (PIM) command line, you must use a JSON file containing the parameters relevant to your installation architecture and pass that file with the `convert` command. The JSON file references your installation architecture, including disaster recovery where applicable.

Important: Creating a JSON file containing installation parameters is not required if you use the guided upgrade process. With the guided process, if conversion is required, PIM notifies you of the requirement and runs the conversion automatically when you confirm that you want to proceed.

Configuration examples

The following examples illustrate how to structure the JSON file for converting different PE configurations.

Parameters for converting an extra-large architecture with disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "replica_host": "pe-xl-core-2.lab1.puppet.vm",
  "replica_postgresql_host": "pe-xl-core-3.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
}
```

Parameters for converting an extra-large architecture without disaster recovery

```
{
  "primary_host": "pe-xl-core-0.lab1.puppet.vm",
  "primary_postgresql_host": "pe-xl-core-1.lab1.puppet.vm",
  "compiler_hosts": [
    "pe-xl-compiler-0.lab1.puppet.vm",
    "pe-xl-compiler-1.lab1.puppet.vm"
  ],
}
```

Parameters for converting a large architecture with disaster recovery

```
{
```

```

"primary_host": "pe-l-core-0.lab1.puppet.vm",
"replica_host": "pe-l-core-2.lab1.puppet.vm",
"compiler_hosts": [
    "pe-l-compiler-0.lab1.puppet.vm",
    "pe-l-compiler-1.lab1.puppet.vm"
],
}

```

Parameters for converting a large architecture without disaster recovery

```

{
    "primary_host": "pe-l-core-0.lab1.puppet.vm",
    "compiler_hosts": [
        "pe-l-compiler-0.lab1.puppet.vm",
        "pe-l-compiler-1.lab1.puppet.vm"
    ],
}

```

Parameters for converting a standard architecture with disaster recovery

```

{
    "primary_host": "pe-core-0.lab1.puppet.vm",
    "replica_host": "pe-core-2.lab1.puppet.vm",
}

```

Parameters for converting a standard architecture without disaster recovery

```

{
    "primary_host": "pe-core-0.lab1.puppet.vm",
}

```

Upgrading agents

Upgrade your agents as new versions of Puppet Enterprise (PE) become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can use a script to upgrade individual nodes.

Important: Before upgrading agents, verify that the primary server and agent software versions are compatible. [Component versions in recent PE releases](#) on page 14 lists which Puppet agent versions are tested and supported for each PE release.

After upgrading, run Puppet on your agents (such as with `puppet agent -t`) as soon as possible to verify that the agents have the correct configuration and your systems are behaving as expected.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

[Installing agents](#) on page 145

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

[Adding and removing agent nodes](#) on page 432

You can add nodes you want to manage with Puppet Enterprise (PE) and remove nodes you no longer need.

[Upgrading Puppet Enterprise](#) on page 182

Upgrade your PE installation as new versions become available.

Upgrade agents using the `puppet_agent` module

You can use the `puppet_agent` module to upgrade multiple *nix, macOS, or Windows agents at one time. The module handles all the latest version-to-version upgrades.

Important: For the most reliable upgrade, use the latest version of the `puppet_agent` module available from the [Forge](#) to upgrade agents. Test the upgrade on a subset of agents, and after you verify the upgrade, upgrade remaining agents.

1. Deploy the `puppet_agent` module using the appropriate method, depending on how your PE installation is configured.
 - If you use Code Manager or r10k to deploy and manage your Puppet code, declare the `puppet-agent` module in the Puppetfile on relevant branches in your control repo. For more information about installing PE modules when you use Code Manager or r10k, see [Managing modules with a Puppetfile](#) on page 768.
 - If you do not use Code Manager or r10k, you can install the `puppet_agent` module by running the following command on your primary server:

```
puppet module install puppetlabs-puppet_agent
```

2. Configure the primary server to download the agent version you want to upgrade to.
 - a) In the PE console, go to **Node groups > PE Infrastructure > PE Master**.
 - b) On the **Classes** tab, enter `pe_repo` in the **Add a new class** field, and select the appropriate repo class from the list of classes.

Repo classes are formatted as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

To use a specific agent version, set the `agent_version` variable using an X.Y.Z format (for example, 7.26.0). If you specify a version in this way, agents **do not** automatically upgrade when you upgrade your primary server.

3. Create an agent upgrade node group.
 - a) Go to **Node groups > Add group**.
 - b) Set the **Parent name** to the name of the classification node group that is the parent of this group, such as **All Nodes**.
 - c) Enter a **Group name** describing the classification node group's role, such as `agent_upgrade`.
 - d) Select the **Environment** your agents are in.
 - e) Do not select the **Environment group** option.
 - f) Click **Add**.
4. Click the link to **Add membership rules, classes, and variables**.
5. On the **Rules** tab, create one or more rules to add the agent nodes you want to upgrade to this group, click **Add Rule**, and then commit changes.

[Dynamically add nodes to a node group](#) on page 443 provides detailed instructions for creating node group rules.
6. Go to the **Classes** tab for the agent node upgrade group, add the `puppet_agent` class, and click **Add class**. You might need to click **Refresh** to update the classifier.

7. Locate the **puppet_agent** class you just added. Select the **package_version** parameter, set the **Value** to the puppet-agent package version you want to install, then commit changes.
If you want to automatically install the same agent version as your primary server, set the **Value** to `auto`. To install a specific version, enter the version number in X.Y.Z format. For example, setting the **Value** to `7.26.0` specifies agent version 7.26.0.
8. If you changed the `prefix` parameter for the **pe_repo** class in the **PE Master** node group, you must communicate this to the agent upgrade node group. To do this, on the agent upgrade node group, set one of the `*_source` parameters for the **puppet_agent** class to `https://<PRIMARY_HOSTNAME>:8140/<PREFIX>`. Go to the **puppet_agent** module's [Forge page](#) for descriptions of the various `*_source` parameters.
9. Run Puppet on the agents you're upgrading, such as: `/opt/puppet/bin/puppet agent -t`

After the Puppet run, you can verify the upgrade with: `/opt/puppetlabs/bin/puppet --version`
Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Upgrade agents using a script

To upgrade the agent on an individual node, you can use a script to upgrade directly from the node. This method relies on a package repository hosted on your primary server.

Tip: If you encounter SSL errors during the upgrade process, make sure the agent node's OpenSSL is updated and matches the primary server's OpenSSL version. Use these commands check OpenSSL versions:

- For the primary server: `/opt/puppetlabs/puppet/bin/openssl version`
- For agent nodes: `openssl version`

Upgrade a *nix agent using a script

You can use a script to upgrade individual *nix agents.

For general information about forming curl commands and authentication in commands, go to [Using example commands](#) on page 28.

1. Configure the primary server to download the agent version you want to upgrade to.
 - a) In the PE console, go to **Node groups > PE Infrastructure > PE Master**.
 - b) On the **Classes** tab, enter `pe_repo` in the **Add a new class** field, and select the appropriate repo class from the list of classes.

Repo classes are formatted as `pe_repo::platform:<AGENT_OS_VERSION_ARCHITECTURE>`. To use a specific agent version, set the `agent_version` variable using an X.Y.Z format (for example, `7.26.0`). If you specify a version in this way, agents **do not** automatically upgrade when you upgrade your primary server.

 - c) Click **Add class** and commit changes.
 - d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`
A new agent package repo is created at `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.
2. SSH into the agent node you want to upgrade.
3. Run the upgrade script command:

```
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8140/packages/current/
install.bash"

curl --cacert "$cacert" "$uri" | sudo bash
```

PE services restart automatically after upgrade.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Upgrade a Windows agent using a script

You can use a script to upgrade individual Windows agents.



CAUTION: For Windows, this method is riskier than when you [Upgrade agents using the puppet_agent module](#) on page 197, because you must manually perform actions and verifications that the `puppet_agent` 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, open PowerShell as an administrator and 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. Run `puppet agent -t` and verify that Puppet runs succeed.
4. Restart the Puppet service and the PXP agent service.

Upgrade agents without internet access

In situations where your primary and agents are airgapped, the primary server can't download the package. Therefore, you have to download the agent tarball from an internet-connected system, prepare the airgapped primary server to serve up the agent package to your agents, and then run the upgrade script on your agents.

1. [Download](#) the appropriate agent tarball.

If you are installing an agent version that is different from your primary server, make sure you download the agent tarball corresponding to the `agent_version` parameter for the node's platform, as explained in [Setting agent versions](#) on page 200.

2. On your primary server, copy the agent tarball to the appropriate agent package directory at: `/opt/puppetlabs/server/data/staging/pe_repo-puppet-agent-<AGENT_VERSION>`

3. Declare the agent architecture class in the **PE Master** node group:

a) In the PE console, go to **Node groups > PE Infrastructure > PE Master**.

b) On the **Classes** tab, enter `pe_repo` in the **Add a new class** field, and select the appropriate repo class from the list of classes.

Repo classes are formatted as `pe_repo::platform::<AGENT_OS_VERSION_ARCHITECTURE>`.

To use a specific agent version, set the `agent_version` variable using an X.Y.Z format (for example, 7.26.0). If you specify a version in this way, agents **do not** automatically upgrade when you upgrade your primary server.

c) Click **Add class** and commit changes.

d) On your primary server, run Puppet to configure the newly assigned class: `puppet agent -t`

A new agent package repo is created at `/opt/puppetlabs/server/data/packages/public/<PE VERSION>/<PLATFORM>/`.

4. SSH into, or physically log on to, the agent node you want to upgrade.
5. Run the upgrade script command:

```
cacert="$(puppet config print localcacert)"
uri="https://$(puppet config print server):8140/packages/current/
install.bash"

curl --cacert "$cacert" "$uri" | sudo bash
```

6. Repeat these steps to upgrade additional agents.

Setting agent versions

Usually, you want your agent nodes to run the same agent version as the primary server; however, if absolutely necessary, agent nodes can run a different, but compatible, version.

Important: Make sure the primary server and agent versions are compatible. [Component versions in recent PE releases](#) on page 14 lists which Puppet agent versions are tested and supported for each PE release.

If you [Upgrade agents using the puppet_agent module](#) on page 197, you specify the agent version by setting the `package_version` parameter on the agent upgrade node group. You can define a specific version or set this to `auto`, if you want your agents to always run the same version as your primary server. When set to `auto`, agent nodes automatically upgrade themselves on their first Puppet run after a primary server upgrade. You can also set the `package_version` parameter for the `puppet_agent` class in the `puppet_agent` module's configuration.

The agent version can be specified on a platform-by-platform basis by the `agent_version` parameter of any `pe_repo::platform` classes in the **PE Master** node group (at **Node Groups > PE Master > Classes**). If your nodes run on various platforms, you must set the `agent_version` on each `pe_repo` class that you want to use a specific agent version. For example, you can specify different versions for 32-bit and 64-bit Windows agents.



CAUTION: Setting `agent_version` blocks upgrades. Setting this parameter is only recommended in specific scenarios with strong justification for doing so.

Never set `agent_version` for infrastructure nodes. Critical failures can occur if all your infrastructure nodes, including the primary server, compilers, and replicas, aren't running the same agent version.

When you install or upgrade agent nodes, the agent install script looks at the node's platform class and installs the specified agent version. If you don't specify a version for a platform, the script installs the default version packaged with your current version of PE. If you specified an older version for your agent platforms, you could upgrade your primary server while maintaining an older agent version on your agent nodes. Similarly, if you specified a newer version for your agent platforms, your agent nodes would run a newer agent version than your primary server.



CAUTION:

The primary server's agent version must match the agent version on other infrastructure nodes, including compilers and replicas, otherwise your primary server won't compile catalogs for those nodes. Not compiling catalogs is a critical failure. Never set `agent_version` on any infrastructure node (including the primary server, compilers, and replicas).

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Migrate PE

As an alternative to upgrading, you can *migrate* your PE installation. Migrating results in little or no downtime, but it requires additional system resources because you must configure a new primary server.

If you have a standard or large installation, you can implement the following migration process:

Migrate your installation

To migrate your installation, create a new primary server. Then, on the new primary server, restore your existing installation from a backup and proceed with upgrading PE.

Before you begin

Review the [upgrade cautions](#) for major changes to architecture and infrastructure components which might affect your upgrade.

1. [Back up](#) your existing installation.
2. [Install](#) your current PE version on a new primary server node.
3. [Restore](#) your installation on the new primary server.
4. [Upgrade](#) your new primary server to the latest PE version.

Configuring Puppet Enterprise

- [Tune infrastructure nodes](#) on page 203

Use these guidelines to configure your Puppet Enterprise (PE) installation to maximize use of available system resources (CPU and RAM).

- [How to configure PE](#) on page 211

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. 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.

- [Configure Puppet Server](#) on page 215

If needed, you can configure Puppet Server settings to optimize your Puppet Enterprise (PE) installation.

- [Configure PuppetDB](#) on page 220

If needed, you can configure PuppetDB settings to optimize your Puppet Enterprise (PE) installation.

- [Configure security settings](#) on page 222

Configure these security settings to ensure your Puppet Enterprise (PE) environment is secure.

- [Configure proxies](#) on page 228

If you have components with limited (or no) internet access, you can configure proxies at various points in your infrastructure, depending on your connectivity limitations.

- [Configure the console](#) on page 230

After installing Puppet Enterprise (PE), you can change product settings to customize the PE console's behavior. You can configure many of these settings directly in the console.

- [Configure orchestration](#) on page 235

After installing PE, you can change some default settings to further configure the orchestrator and `pe-orchestration-services`.

- [Configure ulimit](#) on page 240

As your infrastructure grows and you use Puppet Enterprise (PE) to manage more agents, you might need to increase the number of allowed file handles per client.

- [Analytics data collection](#) on page 241

Some components automatically collect data about how you use Puppet Enterprise (PE). You can opt out of this data collection during or after installing PE.

- [Static catalogs](#) on page 245

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. Puppet Enterprise (PE) primary servers typically compile catalogs from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata specifying the desired state of any file resources containing `source` attributes pointing to `puppet:///` locations on a node.

Related information

[Configure Code Manager](#) on page 778

To configure Code Manager you must enable Code Manager in Puppet Enterprise (PE), set up authentication, and test the connection between the control repository and Code Manager.

[Configuring patch management](#) on page 575

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.

[Configuring disaster recovery](#) on page 248

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.

[About the `pe_status_check` module](#) on page 400

The `pe_status_check` module can alert you when your Puppet Enterprise (PE) installation is not in an ideal state, based on preset indicators, and describe how you can resolve or improve the detected issue.

Tune infrastructure nodes

Use these guidelines to configure your Puppet Enterprise (PE) installation to maximize use of available system resources (CPU and RAM).

PE includes of multiple services running on one or more infrastructure hosts. Services running on the same host share the host's resources. You can configure each service's settings to maximize use of system resources and optimize performance.

Each service's default settings are conservative, and your optimal settings depend of the complexity and scale of your infrastructure.

Configure these settings after you install PE, upgrade PE, or make changes to infrastructure hosts (such as changing existing hosts' system resources, adding new hosts, or adding or changing compilers).

Related information

[Hardware requirements](#) on page 96

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.

Hardware category	Setting	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)	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
	Recommended	2	1024	192	1	819	655	819	1	1638
	With legacy compilers	2	1024	192	2	1228	655	819	1	1638
6 cores, 10 GB RAM	Default	4	2048	512	3	256	256	704	1	1488
	Recommended	3	2304	288	1	1024	819	1024	1	2048
	With legacy compilers	2	1536	192	3	1536	819	1024	1	2048
8 cores, 12 GB RAM	Default	4	2048	512	4	256	256	704	1	2000
	Recommended	3	2304	288	2	1228	983	1228	1	2457
	With legacy compilers	3	2304	288	4	1843	983	1228	1	2457

Hardware	Setting category	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)	JRuby max active instances	Shared buffers (MB)	Work memory (MB)	
10 cores, 16 GB RAM	Default	4	2048	512	5	256	256	704	1	3024	4
	Recommended	5	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	8	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	9	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.

Hardware	Setting category	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.

Hardware	Setting category	Puppet Server		
		JRuby max active instances	Java heap (MB)	Reserved code cache (MB)
4 cores, 8 GB RAM	Default	3	2048	512
	Recommended	3	1536	288
6 cores, 12 GB RAM	Default	4	2048	512
	Recommended	5	3840	480

The `puppet infrastructure tune` command

The `puppet infrastructure tune` command outputs optimized settings for Puppet Enterprise (PE) services based on recommended guidelines.

Running `puppet infrastructure tune` queries PuppetDB to identify processor and memory facts about your infrastructure hosts. The command outputs settings in YAML format for you to use in Hiera.

This command is compatible with most standard PE configurations, including those with compilers, a replica, or standalone PostgreSQL.

You must run this command on your primary server as root. Using `sudo` for elevated privileges is not sufficient. Instead, start a root session by running `sudo su -`, and then run the `puppet infrastructure` command.

These options are commonly used with the `puppet infrastructure tune` command:

- `--current` outputs existing tuning settings from the PE console and Hiera. This option also identifies duplicate settings declared in both the console and Hiera
- `--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, which are identical on several nodes, separately from node-specific settings.

For more information about the `tune` command, run `puppet infrastructure tune --help`.

Restriction: The `puppet infrastructure tune` command fails if `environmentpath` (in your `puppet.conf` file) is set to multiple environments. Comment out this setting before running this command. For details about this setting, refer to [environmentpath in the open source Puppet documentation](#).

Related information

[RAM per JRuby](#) on page 205

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server.

Tuning parameters

Configure tuning parameters to customize your PE service settings for optimum performance and hardware resource utilization.

Specify tuning parameters in Hiera for the best scalability and consistency. You can learn [About Hiera](#) in the Puppet documentation.

If you must use the PE console, add the parameter to the appropriate infrastructure node group using one of the following methods:

- 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.

[How to configure PE](#) on page 211 explains the different ways you can configure PE parameters.

RAM per JRuby

The `puppetserver_ram_per_jruby` setting determines how much RAM is allocated to each JRuby instance in Puppet Server.

You might need to change this setting if you have complex Hiera code, many environments or modules, or large reports.

Tip: If your PuppetDB service runs on a compiler, this is a good starting point for tuning your infrastructure, because this value is factored into several other parameters, including [JRuby max active instances](#) on page 206 and [Java heap](#) on page 207 allocation on compilers running PuppetDB.

Console node group

PE Master

Parameter

`puppet_enterprise::puppetserver_ram_per_jruby`

Default value

512 MB

Accepted values

An integer representing a number of MB

How to calculate

You can usually achieve good performance by allocating around 2 GB per JRuby.

If 2 GB is inadequate, it might help to [Change the environment_timeout setting](#) on page 217.

JRuby max active instances

The `jruby_max_active_instances` setting can be set in multiple places. It controls the maximum number of JRuby instances to allow on the Puppet Server and how many plans can run concurrently in the orchestrator.

Puppet Server `jruby_max_active_instances`

Console node group

If Puppet Server runs on the primary server: PE Master

If the PuppetDB service runs on compilers: PE Compiler

Parameter

`puppet_enterprise::master::puppetserver::jruby_max_active_instances`

Tip: This parameter is the same as the `max_active_instances` parameter in the [pe-puppet-server.conf settings](#) on page 219 and in open source Puppet.

Default value

If Puppet Server runs on the primary server, the default value is the number of CPUs minus 1. The minimum is 1, and the maximum is 4.

If the PuppetDB service runs on compilers, the default value is the number of CPUs multiplied by 0.75. The minimum is 1, and the maximum is 24.

Accepted values

An integer representing a number of JRuby instances

How to calculate

As a conservative estimate, one JRuby process uses approximately 512 MB of RAM. For most installations, four JRuby instances are adequate.

Important: Because increasing the maximum number of JRuby instances also increases the amount of RAM used by Puppet Server, make sure to proportionally scale the Puppet Server [Java heap](#) on page 207 size (`java_args`). For example, if you set `jruby_max_active_instances` to 4, set Puppet Server's `java_args` to at least 2 GB.

Orchestrator `jruby_max_active_instances`

Running a plan consumes one JRuby instance. If a plan calls other plans, the nested plans use the parent plan's JRuby instance. JRuby instances are deallocated once a plan finishes running, and tasks are not affected by JRuby availability.

Console node group

PE Orchestrator

Parameter

```
puppet_enterprise::profile::orchestrator::jruby_max_active_instances
```

Default value

The default value is the orchestrator heap size (`java_args`) divided by 1024. The minimum is 1.

Accepted values

An integer representing a number of JRuby instances

How to calculate

Because the `jruby_max_active_instances` default value is derived from the orchestrator heap size (`java_args`), changing the orchestrator heap size automatically changes the number of JRuby instances available to the orchestrator. For example, setting the orchestrator heap size to 5120 MB allows up to five JRuby instances (or plans) to run concurrently.

If you notice poor performance while running plans, increase the orchestrator [Java heap](#) on page 207 size instead of `jruby_max_active_instances`. However, keep in mind that allowing too many JRuby instances can reduce system performance, especially if your plans use a lot of memory.

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.

Console node group

PE Master

Parameter

```
puppet_enterprise::master::puppetserver::jruby_max_requests_per_instance
```

Tip: This parameter is the same as the `max_requests_per_instance` parameter in the [pe-puppet-server.conf settings](#) on page 219 and in open source Puppet.

Default value

100000

Accepted values

An integer representing a number of HTTP requests

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. Therefore, set this parameter to get a new interpreter no more than once every few hours.

Requests are balanced across multiple interpreters running concurrently, so the lifespan of each interpreter varies.

Java heap

The `java_args` settings specify *heap size*, which is the amount of memory that each Java process can request from the operating system. You can specify a heap size for each PE service that uses Java, including Puppet Server, PuppetDB, the console, and the orchestrator

Heap size is declared as a JSON hash containing a maximum (`Xmx`) and minimum (`Xms`) value. Usually, the maximum and minimum are the same so that the heap size is fixed, for example:

```
{ 'Xmx' => '2048m', 'Xms' => '2048m' }
```

Puppet Server Java heap

Console node group: PE Master or PE Compiler

Parameter: `puppet_enterprise::profile::master::java_args`

Tip: `puppet_enterprise::master::java_args` and `puppet_enterprise::master::puppetserver::java_args` are the same, because `profile::master` filters down to `master`, which filters down to `master::puppetserver`.

Default value: 2 GB

PuppetDB Java heap

Console node group: If the PuppetDB service runs on compilers, set this parameter on the PE Compiler node group. Otherwise, set this parameter on the PE PuppetDB node group.

Parameter: `puppet_enterprise::profile::puppetdb`

Default value: 256 MB

Console services Java heap

Console node group: PE Console

Parameter: `puppet_enterprise::profile::console`

Default value: 256 MB

Orchestrator Java heap

Console node group: PE Orchestrator

Parameter: `puppet_enterprise::profile::orchestrator`

Default value: 704 MB

Related information

[Orchestrator and pe-orchestration-services parameters](#) on page 236

These are some optional parameters you can use to configure the behavior of the orchestrator and the `pe-orchestration-services` service.

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.

Console node group

If the PuppetDB service runs on compilers, set this parameter on the PE Compiler node group. Otherwise, set this parameter on the PE Master node group.

Parameter

`puppet_enterprise::master::puppetserver::reserved_code_cache`

Default value

If Puppet Server runs on your primary server: If total RAM is less than 2 GB, then the Java default is used. Otherwise, the default value is 512 MB.

If the PuppetDB service runs on compilers: The default value is the number of JRuby instances multiplied by 128 MB. The minimum is 128 MB, and the maximum is 2048 MB.

Accepted values

An integer representing a number of MB

How to calculate

JRuby requires an estimated 128 MB of cache space for each instance. To determine the minimum amount of space needed multiple the number of JRuby instances by 128 MB.

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 one command at a time.

Console node group

If the PuppetDB service runs on compilers, set this parameter on the PE Compiler node group. Otherwise, set this parameter on the PE PuppetDB node group.

Parameter

```
puppet_enterprise::puppetdb::command_processing_threads
```

Default value

If the PuppetDB service runs on compilers, the default value is the number of CPUs multiplied by 0.25 (with a minimum of 1 and a maximum of 3).

Otherwise, the default value is the number of CPUs multiplied by 0.5 (with a minimum of 1).

Accepted values

An integer representing a number of threads.

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.

Don't allocate all of your CPU cores to command processing, because this can starve other PuppetDB subsystems of resources and decrease throughput.

Related information

[PuppetDB parameters](#) on page 139

Use these parameters to configure and tune PuppetDB.

[Configure PuppetDB](#) on page 220

If needed, you can configure PuppetDB settings to optimize your Puppet Enterprise (PE) installation.

PostgreSQL max connections

The `max_connections` setting determines the maximum number of concurrent connections allowed to the PE-PostgreSQL server. It should be configured to accommodate all infrastructure nodes running PuppetDB.

Console node group

PE Database

Parameter

```
puppet_enterprise::profile::database::max_connections
```

Default value

400

Accepted values

An integer representing the number of concurrent connections allowed. The minimum is 200.

How to calculate

Set the `max_connections` parameter to a number greater than the sum of read and write connections across all PuppetDB instances in your PE installation, including compilers and the primary server. The connection count from each instance should equal (`command_processing_threads * 2`) + number of JRuby instances. Rule out any underlying performance issues prior to adjusting `max_connections`.

PostgreSQL shared buffers

The `shared_buffers` setting specifies the amount of memory the PE-PostgreSQL server uses for shared memory buffers.

Console node group

PE Database

Parameter

```
puppet_enterprise::profile::database::shared_buffers
```

Default value

The available RAM multiplied by 0.25, with a minimum of 32 MB and a maximum of 4096 MB

Accepted values

An integer representing a number of MB

How to calculate

The default value is suitable for most installations, but console performance might improve if you increase `shared_buffers` up to 40% of available RAM.

PostgreSQL working memory

The `work_mem` setting specifies the maximum amount of memory used for queries before writing to temporary files.

Console node group

PE Database

Parameter

```
puppet_enterprise::profile::database::work_mem
```

Default value

Based on the following calculation:

```
(Available RAM / 1024 / 8) + 0.5
```

The minimum is 4 MB, and the maximum is 16 MB.

Accepted values

An integer representing a number of MB

PostgreSQL WAL disk space

The `max_slot_wal_keep_size` setting specifies the maximum allocated WAL disk space for each replication slot. This prevents the `pg_wal` directory from growing infinitely.

If you have set up disaster recovery, this setting prevents an unreachable replica from consuming all of your primary server's disk space when the PE-PostgreSQL service on the primary server attempts to retain change logs that the replica hasn't acknowledged.

If your replica is offline long enough to reach the `max_slot_wal_keep_size` value, replication slots are dropped to allow the primary server to continue functioning normally. When the replica comes back online, you'll know replication slots were dropped if `puppet infra` status returns a message that replication is inactive for PostgreSQL's status. To restore PostgreSQL replication, run `puppet infra reinitialize replica` on your replica.

Console node group

PE Database

Parameter

```
puppet_enterprise::profile::database::max_slot_wal_keep_size
```

Default value

12288 MB (twice the size of the `max_wal_size` parameter)

Important: If you don't have enough disk space for the default setting, you must adjust this value.

Accepted values

An integer representing a number of MB

Related information

[Configuring disaster recovery](#) on page 248

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.

How to configure PE

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. 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 (which are documented in the [Configuration Reference](#)). However, the default values for PE might differ from the default values for Puppet. Some examples of settings that have different defaults in PE include `disable18n`, `environment_timeout`, `always_retry_plugins`, and the Puppet Server JRuby `max-active-instances` settings. To verify PE's configuration defaults, check the `puppet.conf` file after installation.

There are three ways to configure PE settings:

- [Configure settings in the PE console](#) on page 211
- [Configure settings with Hiera](#) on page 212
- [Configure settings in pe.conf](#) on page 213

For consistency, it is important to always configure settings in the same way, unless a situation calls for you to use a specific method. For example, if you choose to configure settings in the PE console, then always configure settings in the console, unless a specific setting requires using Hiera or editing `pe.conf`.

This page provides generic instructions for configuring PE settings. You'll find information about specific settings in other [Configuring Puppet Enterprise](#) on page 202 topics and throughout the PE documentation.

Configure settings in the PE console

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

Changes you make in the console override your Hiera data and data in `pe.conf`. It is best to use the console when you want to:

- Change parameters in profile classes starting with `puppet_enterprise::profile`.
- Add parameters to PE-managed configuration files.
- Set parameters that configure at runtime.

To change settings in the console you can [Set configuration data](#) on page 211 or [Set parameters](#) on page 212.

Related information

[Preconfigured node groups](#) on page 455

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 in 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 permission to define or edit 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 enable data editing in the console for both your primary server and replica.

1. In the console, click **Node groups** and select the node group that you want to add configuration data to.
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: If necessary, change the parameter's default **Value**.

Related information

[Enable data editing in the console](#) on page 235

In new Puppet Enterprise (PE) installations, you can, by default, edit configuration data in the console. If you upgraded from an earlier PE version where you hadn't already enabled editing of configuration data, you must use Hiera to manually enable **Classifier Configuration Data**.

Set parameters

Parameters are declared resource-style, which means you can use them to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

Important: You can structure parameters as JSON, but, if they can't be parsed as JSON, they're treated as strings.

1. In the console, click **Node groups** and select the node group you want to add a parameter to.
2. On the **Classes** tab, select the class you want to modify, and select the **Parameter** you want to add. The **Parameter** list shows all parameters available for the selected 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: If necessary, change the parameter's default **Value**.

Related information

[Tips for specifying parameter and variable values](#) on page 447

Parameters and variables can be structured as JSON, but, if they can't be parsed as JSON, they're treated as strings.

Configure settings with Hiera

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Before you begin

[Separating data \(Hiera\)](#) in the Puppet documentation explains more about how to use Hiera and what you can configure in Hiera.

Changes to PE configuration settings in Hiera override configuration settings in `pe.conf`, but not those set in the PE console. However, settings declared in the console override Hiera data. 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 in Hiera:

1. Open a Hiera data file in a text editor.

The default location for Hiera data files on *nix systems is:

```
/etc/puppetlabs/code/environments/<ENVIRONMENT>/data/common.yaml
```

On Windows systems, it is:

```
%CommonAppData%\PuppetLabs\code\environments\<ENVIRONMENT>\data\common.yaml
```

Tip: The `datadir` setting in the `hiera.yaml` configuration file changes the Hiera data file location. You can also change the common data file path in the `hierarchy` section of the `hiera.yaml` file. If you changed either of these settings, you'll find the default Hiera data files in your customized location.

2. Add your new parameter to the Hiera data file.

For example, the following declaration increases sets number of seconds before a node is considered unresponsive to 4000, whereas the default setting is 3600 seconds:

```
puppet_enterprise::console_services::no_longer_reporting_cutoff: 4000
```

3. Save the file and run `puppet agent -t` to compile the changes.

Related information

[Preconfigured node groups](#) on page 455

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 in `pe.conf` when you want to:

- Access settings during installation.
- Configure for high availability.

To configure settings in `pe.conf`:

1. On your primary server, open the `pe.conf` file in a text editor. The file is located at:

```
/etc/puppetlabs/enterprise/conf.d/pe.conf
```

2. Add the parameter and value you want to set.

For example, this declaration changes the proxy location in your PE repo:

```
pe_repo::http_proxy_host: "proxy.example.vlan"
```

3. Save the file and run `puppet agent -t`

Tip: If you had stopped any PE services, run `puppet infrastructure configure` instead of `puppet agent -t`.

Configuration file syntax

Puppet supports two formats for configuration files: valid JSON and Human-Optimized Config Object Notation (HOCON), which is a JSON superset. We've provided these syntax examples to guide you when you're writing configuration files.

For details about HOCON itself, refer to the [HOCON documentation](#).

Brackets

JSON example with brackets:

```
{
  "authorization": {
    "version": 1
  }
}
```

In HOCON, you can omit the brackets ({ }) around the root object. For example:

```
"authorization": {
  "version": 1
}
```

Quotation marks

With JSON, wrap keys in double quotes. Quotation marks around values depends on the value type, such as an integer or string. For example:

```
"authorization": {
  "version": 1
}
```

In HOCON, double quotes around keys and string values are usually optional. However, double quotes are required if the string contains any of these characters: *, ^, +, :, or =

For example:

```
authorization: {
  version: 1
}
```

Commas

In JSON, use commas to separate items in a map or array.

JSON map example:

```
rbac: {
  password-reset-expiration: 24,
  session-timeout: 60,
  failed-attempts-lockout: 10,
}
```

JSON array example:

```
http-client: {
  ssl-protocols: [TLSv1, TLSv1.1, TLSv1.2, TLSv1.3]
}
```

When writing a map or array in HOCON, you can use a new line instead of a comma.

HOCON map example:

```
rbac: {
  password-reset-expiration: 24
  session-timeout: 60
  failed-attempts-lockout: 10
```

```
}
```

HOCON array example:

```
http-client: {
  ssl-protocols: [
    TLSv1
    TLSv1.1
    TLSv1.2
  ]
}
```

Comments

JSON does not support comments.

In HOCON, you can use // or # to delineate comments. Inline comments are supported. For example:

```
authorization: {
  version: 1
  rules: [
    {
      # Allow nodes to retrieve their own catalog
      match-request: {
        path: "^/puppet/v3/catalog/([^\n]+)$"
        type: regex
        method: [get, post]
      }
    }
  ]
}
```

Configure Puppet Server

If needed, you can configure Puppet Server settings to optimize your Puppet Enterprise (PE) installation.

Related information

[Puppet Server reserved code cache](#) on page 208

The `reserved_code_cache` setting specifies the maximum space available to store the Puppet Server code cache during catalog compilation.

Set the Ruby load path

The `ruby_load_path` setting determines where Puppet Server finds components such as Puppet and Facter.

The default setting is:

```
$puppetserver_jruby_puppet_ruby_load_path = [ \
  '/opt/puppetlabs/puppet/lib/ruby/vendor_ruby', \
  '/opt/puppetlabs/puppet/cache/lib' ]
```

Important: If you change the `libdir` you must also change the `vardir`.

You can change the load path array in `pe.conf`.

1. On your primary server, open the `pe.conf` file in a text editor.

- Add the following parameter and specify the new load path array:

```
puppet_enterprise::master::puppetserver::puppetserver_jruby_puppet_ruby_load_path:  
  \  
  [ '<PATH1>', '<PATH2>' ]
```

- Save the file and run: `puppet agent -t`

Enable JRuby multithreading

The `jruby_puppet_multithreaded` setting enables multithreaded mode, which uses a single JRuby instance to process requests (such as catalog compiles) concurrently.

By default, multithreading is disabled (set to `false`).



CAUTION: Multithreaded mode is an experimental feature which might experience breaking changes in future releases. Test this feature in a non-production environment before enabling it in production.

You can use Hiera to toggle multithreaded mode.

- Open your default Hiera `.yaml` file in a text editor.

Tip: For information about Hiera data files, including file paths, refer to [Configure settings with Hiera](#) on page 212.

- Add the `jruby_puppet_multithreaded` parameter, and set it to either `true` (enabled) or `false` (disabled). For example:

```
puppet_enterprise::master::puppetserver::jruby_puppet_multithreaded: true
```

- Save the file and run `puppet agent -t` to compile the changes.

Use cached data when updating classes

The `environment_class_cache_enabled` setting specifies whether cached data is used when updating classes in the Puppet Enterprise (PE) console. When `true`, Puppet Server uses file sync when refreshing classes, which provides improved performance.

The default value for `environment_class_cache_enabled` depends on whether you use Code Manager:

- Without Code Manager, the default value is `false` (disabled).
- With Code Manager, the default value is `true` (enabled).

When enabled, file sync automatically clears the environment cache in the background. This means you don't have to manually clear the environment cache if you're using Code Manager.

Important: If you're not using Code Manager and you set `environment_class_cache_enabled` to `true`, you must make sure your chosen code deployment method (such as r10k) clears the environment cache when it completes code deployments. If the environment cache isn't cleared, the Node Classifier service doesn't receive new class information until the Puppet Server service is restarted.

You can use Hiera to toggle the `environment_class_cache_enabled` setting.

- Open your default Hiera `.yaml` file in a text editor.

Tip: For information about Hiera data files, including file paths, refer to [Configure settings with Hiera](#) on page 212.

- Add the `jruby_environment_class_cache_enabled` parameter, and set it to either `true` (enabled) or `false` (disabled). For example:

```
puppet_enterprise::master::puppetserver::jruby_environment_class_cache_enabled:  
  true
```

- Save the file and run `puppet agent -t` to compile the changes.

Change the environment_timeout setting

The `environment_timeout` setting controls if and how long the primary server caches environment data. Environment caching can reduce your Puppet Server's CPU usage, but longer cache times extend the amount of time it takes for environments to reflect changes to their Puppet code.

The `environment_timeout` parameter accepts these values:

- No caching: `0`
- Retain environment data caches indefinitely: `unlimited`
- Cache environments for a specified length of time after their last use: Any length of time, such as `5m`

By default, `environment_timeout` is set to `0`. When you [Enable Code Manager](#) on page 779, `environment_timeout` is set to `5m`. However, if you set `code_manager_auto_configure` to `true` in your [Code Manager settings](#) on page 782, then `environment_timeout` is automatically set to `unlimited`.

Tip: Setting `environment_timeout` to `0` taxes your primary server's performance but makes it easy for new users to deploy updated Puppet code. Once your code deployment process is mature (or after enabling Code Manager), we recommend changing this setting to `unlimited`.

Refer to the Puppet documentation for more information [About environments](#), including [Environment limitations](#), such as leakage and resource conflicts.

The following steps explain how to change the `environment_timeout` setting in `pe.conf`. You can also change this setting in the PE console in the **PE Master** node group. For instructions on changing settings in the console, refer to [Configure settings in the PE console](#) on page 211.

- On your primary server, open the `pe.conf` file in a text editor.
- Add the `environment_timeout` parameter and the desired value. For example:

```
puppet_enterprise::master::environment_timeout: 0
```

- Save the file and run: `puppet agent -t`

Populate the puppet-admin certificate allowlist

Use `pe.conf` to add trusted certificates to the `puppet-admin` certificate allowlist.

- On your primary server, open the `pe.conf` file in a text editor.
- Add the `puppet_admin_certs` parameter and string-formatted certnames to the `pe.conf` file. For example:

```
puppet_enterprise::master::puppetserver::puppet_admin_certs: '<CERTIFICATE_NAME>'
```

- Save the file and run: `puppet agent -t`

Disable software update monitoring

The Puppet Server (`pe-puppetserver`) service checks for updates when it starts, restarts, and every 24 hours while running. You can disable these automatic software update checks.

To check if a software update is available, the `pe-puppetserver` service sends the following basic, anonymous information to our servers at Puppet by Perforce:

- Product name
- Puppet Server version
- IP address
- Data collection timestamp

You can turn off automatic software update monitoring in the Puppet Enterprise (PE) console.

1. In the PE console, go to **Node groups** and select the **PE Master** node group.
2. On the **Classes** tab, find the `puppet_enterprise::profile::master` class.
3. Add the `check_for_updates` parameter and change the value to `false`.
4. Click **Add parameter** and commit changes.
5. On the nodes that host your primary server and console, run Puppet.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Puppet Server configuration files

At startup, Puppet Server reads all `.conf` files in the `conf.d` directory, which is located at `/etc/puppetlabs/puppetserver/conf.d`.

The `conf.d` directory contains these files:

File name	Description
<code>auth.conf</code>	Contains authentication rules and settings for agents and API endpoint access. You can learn more about auth.conf in the Puppet documentation.
<code>global.conf</code>	Contains global configuration settings for Puppet Server, including logging settings. You can learn more about global.conf in the Puppet documentation.
<code>metrics.conf</code>	Contains settings for Puppet Server metrics services. You can learn more about metrics.conf in the Puppet documentation.
<code>pe-puppet-server.conf</code>	Contains Puppet Server settings specific to Puppet Enterprise. Refer to pe-puppet-server.conf settings on page 219 for details about each setting.
<code>webserver.conf</code>	Contains SSL service configuration settings. You can learn more about webserver.conf in the Puppet documentation.
<code>ca.conf</code>	Deprecated. Contained rules for Certificate Authority services, but has been superseded by <code>webserver.conf</code> and <code>auth.conf</code> .

Additional files, such as `code-manager.conf` might exist depending on how you use PE.

Related information

[View and manage Puppet Server metrics](#) on page 401

Puppet Server tracks performance and status metrics you can use to monitor 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. All the settings are wrapped in a `jruby-puppet` section.

enable-file-sync-locking or `file_sync::file_sync_locking_enabled`

Controls whether the file sync client locks the JRuby pool (and, by extension, most requests to Puppet Server) while deploying Puppet code.

Default: `true`



CAUTION: We **do not** recommend changing the `enable-file-sync-locking` setting. Instead, enable [Lockless code deploys](#) on page 783, which allow the file sync client to update code into versioned code directories without blocking Puppet Server requests or overwriting the live code directory.

gem-home

Determines where JRuby looks for gems. This is also used by the `puppetserver gem` command line tool.

Default: `'/opt/puppetlabs/puppet/cache/jruby-gems'`

jruby_max_active_instances

Controls the maximum number of JRuby instances to allow on the Puppet Server.

Default: `4`

For more information, refer to [JRuby max active instances](#) on page 206.

max_requests_per_instance

Sets the maximum number of requests allowed for each JRuby interpreter instance before it is killed.

Default: `100000`

For more information, refer to [JRuby max requests per instance](#) on page 207.

max-queued-requests

Sets the maximum number of requests allowed to be queued waiting to borrow from the JRuby pool.

This setting is optional and defaults to 0 (unlimited). If changed, you must specify a positive integer.

After reaching the limit, all new requests receive a `503 Service Unavailable` response until the queue drops below the limit. If the `max-retry-delay` setting is also set to a positive integer, then the `503` response includes a random sleep time after which the client can retry the request.

Don't use this setting if your managed infrastructure includes multiple agents older than Puppet 5.3. Because older agents treat `503` responses as failures, a thundering herd problem occurs when the agents schedule their next runs at the same time.

max-retry-delay

Sets the maximum number of seconds allowed for the random sleep time set when the `max-queued-requests` limit is exceeded. The random sleep time is returned as a `Retry-After` header on the `503` response for each rejected request.

Default: `1800` seconds

If `max-queued-requests` is 0, there is no limit to the number of queued requests and, therefore, the `max-retry-delay` is irrelevant.

pre-commit-hook-commands or `puppetserver::pre_commit_hook_commands`

Specify scripts, as an array of strings, that you want the file sync storage server to execute against a repo after receiving a change but before committing and syncing the change across compilers. This is similar to [Configuring post-deployment commands](#) on page 824 for r10k or [Configuring post-environment hooks](#) on page 786 for Code Manager.

The `pe-puppetserver` process, acting as the `pe-puppet` user, executes the scripts in the order supplied. You must supply scripts as absolute paths. Additionally, the `pe-puppet` user must be able to execute the scripts, and the scripts must be able to consume `stdin` (even if the script doesn't do anything with it).

Default: ["/opt/puppetlabs/server/bin/generate-puppet-types.rb"]



CAUTION: Removing the default value disables the `generate-puppet-types.rb` script. Unless you intentionally want to disable this type generation method, make sure to keep the default path in the array.

The output, error, and exit code for these scripts are logged at the trace level of `pe-puppetserver` logs when the exit code is 0. If the exit code is not 0, the codes are logged at the error level.

This setting is managed in PE modules, and you can override it by setting the `puppet_enterprise::master::puppetserver::pre_commit_hook_commands` parameter in Hiera. Make sure to include the default path (for `generate-puppet-types.rb`) to ensure custom type are correctly cached. If you want to disable all pre-commit commands, supply an empty array in Hiera.

puppet-code-repo

Identifies, as a string, the internal name for the Puppet code repo (the `codedir`) that contains all code to sync across compilers (including user-supplied code repos).



CAUTION: Do not change this setting.

Default: 'puppet-code'

ruby-load-path or puppetserver_jruby_puppet_ruby_load_path

Determines where Puppet Server finds components such as Puppet and Facter. The agent's `libdir` path is included by default.

Default: ['/opt/puppetlabs/puppet/lib/ruby/vendor_ruby' , '/opt/puppetlabs/puppet/cache/lib']

For more information, refer to [Set the Ruby load path](#) on page 215.

server-conf-dir

Sets the Puppet configuration directory path.

Default: '/etc/puppetlabs/puppet'

server-var-dir

Sets the Puppet Server variable directory path.

Default: '/opt/puppetlabs/server/data/puppetserver'

Configure PuppetDB

If needed, you can configure PuppetDB settings to optimize your Puppet Enterprise (PE) installation.

We've described some commonly-configured parameters here. For additional settings and information, refer to [Configuring PuppetDB](#) in the Puppet documentation, as well as the other PE documentation listed under **Related information**.

Related information

[PuppetDB parameters](#) on page 139

Use these parameters to configure and tune PuppetDB.

Disable agent run reports

By default, every time Puppet runs, your Puppet Enterprise (PE) primary server generates *agent run reports* and submits them to PuppetDB. You can disable agent run reports.

1. In the PE console, navigate to **Node groups > PE Infrastructure > PE Master**.
2. On the **Classes** tab, add the `puppet_enterprise::profile::master::puppetdb` class.
3. Add the `report_processor_ensure` parameter, and set the value to either:
 - "present": Enable agent run reports
 - "absent": Disable agent run reports
4. Click **Add parameter** and commit changes.
5. On the nodes hosting your primary server and PE console, run Puppet.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Set the deactivated node retention time

Use the `node_purge_ttl` parameter to set the length of time before PE automatically removes deactivated or expired nodes. Once the time limit passes, the nodes and their relevant facts, catalogs, and reports are only removed from PuppetDB. Agent certificates on the Certificate Authority (CA) server are untouched.

1. In the PE console, navigate to **Node groups > PE Infrastructure > PE PuppetDB**.
2. On the **Classes** tab, find the `puppet_enterprise::profile::puppetdb` class.
3. Add the `node_purge_ttl` parameter, and set the value to a string representing the desired retention time. Specify a number along with one of the following suffixes:
 - Days: d
 - Hours: h
 - Minutes: m
 - Seconds: s
 - Milliseconds: ms

For example, to set the purge time to 14 days, set the value to `14d`. For example:

```
puppet_enterprise::profile::puppetdb::node_purge_ttl: '14d'
```

4. Click **Add parameter** and commit changes.
5. On the nodes hosting your primary server and PE console, run Puppet.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Change the PuppetDB user password

The Puppet Enterprise (PE) console uses a database user account to access its PostgreSQL database. Change this database user's password if it is compromised or to comply with your organization's security guidelines.

1. Stop the `pe-puppetdb` service by running:

```
puppet resource service pe-puppetdb ensure=stopped
```

2. On the database server (which, depending on your deployment's architecture, might or might not be the same as the PuppetDB server), use your preferred PostgreSQL administration tool to change the user's password.

With the standard PostgreSQL client, you can do this by running:

```
ALTER USER console PASSWORD '<new password>';
```

3. On the PuppetDB server, open the `database.ini` file located at `/etc/puppetlabs/puppetdb/conf.d/database.ini`, and change the `password` line to reflect the new password.
The `password` line is under either `common` or `production`, depending on your configuration.
4. Save the file and restart the `pe-puppetdb` service on the console server by running:

```
puppet resource service pe-puppetdb ensure=running
```

Exclude facts

Use the `facts_blocklist` parameter to exclude facts from being stored in the PuppetDB database.

For more information, you can read about [facts-blocklist](#) in the Puppet documentation.

You can use Hiera to exclude facts:

1. Open your default Hiera `.yaml` file in a text editor.

Tip: For information about Hiera data files, including file paths, refer to [Configure settings with Hiera](#) on page 212.

2. Add the `facts_blocklist` parameter and a list of names of facts that you want to exclude.

For example, this declaration excludes the `system_uptime_example` and `mountpoints_example` facts:

```
puppet_enterprise::puppetdb::database_ini::facts_blocklist:
  - 'system_uptime_example'
  - 'mountpoints_example'
```

3. Save the file and run `puppet agent -t` to compile the changes.

Configure security settings

Configure these security settings to ensure your Puppet Enterprise (PE) environment is secure.

Related information

[Security and communications](#) on page 12

Puppet Enterprise (PE) services and components use a variety of communication and security protocols.

[FIPS 140-2 enabled PE](#) on page 17

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.

[Managing access](#) on page 268

Role-based access control (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.

[Manage permissions with the acl module](#) on page 470

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.

[Configure puppet-access](#) on page 303

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.

[Accepting the console's certificate](#) on page 266

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.

[Managing patches](#) on page 574

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.

[Secure coding practices for tasks](#) on page 627

Use secure coding practices when you write tasks and help protect your system.

[Forget a replica](#) on page 264

Forgetting a replica removes the replica from classification and database state and purges the node.

Configure cipher suites

Regulatory compliance or other security requirements might require you to change the cipher suites your SSL-enabled PE services use to communicate with other PE components.

Before you begin: Make sure you're using [Compatible ciphers](#) on page 13.

To add or remove cipher suites for different service types, use Hiera to modify the following parameters:

`puppet_enterprise::ssl_cipher_suites`

List IANA-formatted ciphers for all PE Java-based services, which includes PuppetDB, Puppet Server, console services, and the orchestrator.

Format: Array of strings

Example:

```
puppet_enterprise::ssl_cipher_suites:
  - 'TLS_AES_256_GCM_SHA384'
  - 'TLS_AES_128_GCM_SHA256'
  - 'TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256'
  - 'TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384'
```

`puppet_enterprise::ssl_cipher_suites_non_java`

List OpenSSL-formatted ciphers for all PE non-Java services, which includes Bolt Server, ACE Server, and PostgreSQL.

Format: Array of strings

Example:

```
puppet_enterprise::ssl_cipher_suites_non_java:
  - 'ECDHE-ECDSA-AES128-GCM-SHA256'
  - 'ECDHE-ECDSA-AES256-GCM-SHA384'
  - 'ECDHE-RSA-AES128-GCM-SHA256'
  - 'ECDHE-RSA-AES256-GCM-SHA384'
```

`puppet_enterprise::ssl_cipher_suites_browser`

List OpenSSL-formatted ciphers for NGINX. These ciphers are accepted by the PE console in the browser.

Format: Array of strings

Example:

```
puppet_enterprise::ssl_cipher_suites_browser:
  - 'TLS_CHACHA20_POLY1305_SHA256'
  - 'ECDHE-ECDSA-CHACHA20-POLY1305'
  - 'ECDHE-RSA-CHACHA20-POLY1305'
  - 'ECDHE-ECDSA-AES128-GCM-SHA256'
```

Related information

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Configure SSL protocols

You can change what SSL protocols your Puppet Enterprise (PE) infrastructure uses.

Where to configure

In Hiera data files.

In the PE console on the **PE Infrastructure** node group's **Configuration data** tab.

Parameter

```
puppet_enterprise::master::puppetserver::ssl_protocols
```

Format

Array of strings representing SSL protocols.

Example

This declaration enables TSLv1.3 and TSLv1.2:

```
puppet_enterprise::master::puppetserver::ssl_protocols: [ "TLSv1.3",
  "TLSv1.2" ]
```

Default

```
[ "TLSv1.3", "TLSv1.2" ]
```

Note: To comply with security regulations, only version 1.2 and 1.3 of the Transport Layer Security (TLS) protocol are enabled. If necessary, you can manually enable TLSv1 and TSLv1.1.

Related information

[Enable TLSv1](#) on page 845

To comply with security regulations, TLSv1 and TLSv1.1 are disabled by default in 2021.7.z versions of Puppet Enterprise (PE).

[SSL and certificates](#) on page 836

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.

[Use a custom SSL certificate for the console](#) on page 843

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

Configure RBAC and token-based authentication settings

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

RBAC and token authentication settings can be changed in the **PE Infrastructure** node group in the Puppet Enterprise (PE) console or in Hiera data. Settings include:

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.

Default: 10

puppet_enterprise::profile::console::rbac_password_reset_expiration

An integer representing the number of hours that *password reset* tokens are valid.

An administrator generates these token for users to reset their passwords.

Default: 24

puppet_enterprise::profile::console::rbac_session_timeout

An 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

puppet_enterprise::profile::console::rbac_token_auth_lifetime

A string representing the default authentication lifetime for a token.

The value is formatted as string consisting of a number followed by a suffix representing a unit of time: **y** (years), **d** (days), **h** (hours), **m** (minutes), or **s** (seconds).

Important: This value cannot exceed the `rbac_token_maximum_lifetime`.

Default: "1h" (one hour)

puppet_enterprise::profile::console::rbac_token_maximum_lifetime

A string representing the maximum allowable lifetime for all tokens.

The value is formatted as a string consisting of a number followed by a suffix representing a unit of time: **y** (years), **d** (days), **h** (hours), **m** (minutes), or **s** (seconds).

Default: 10y (10 years)

puppet_enterprise::profile::console::rbac_account_expiry_check_minutes

An integer specifying, in minutes, how often the application checks for idle user accounts.

Default: 60 minutes

Important: The `rbac_account_expiry_check_minutes` parameter is ignored if you do not also specify the `rbac_account_expiry_days` parameter.

puppet_enterprise::profile::console::rbac_account_expiry_days

An integer specifying, in days, the duration before an inactive user account expires.

The default value is undefined. To activate this feature, specify a positive integer of 1 or greater.

When non-superusers don't log into the console during the specified period, their user status changes to revoked. This also applies to new accounts – The inactivity timer starts once the account is created.

puppet_enterprise::profile::console::ldap_sync_period_seconds

An integer specifying, in seconds, the interval at which LDAP group membership associations are synchronized.

The default value is 1800 (30 minutes).

This synchronization refreshes group membership for every LDAP user in the system, regardless of the last time the user logged in. If a user is no longer present in LDAP or has no group bindings, then all user-group associations are removed from the user and all of the user's known tokens are revoked; however, the user object itself is not removed. If a user is present in LDAP and has group bindings, this synchronization updates the user's group membership, user name, and descriptions (if this data had changed).

To disable automatic synchronization, set the value to 0 or a negative integer. When disabled, user names, descriptions, and group membership only refresh when users log in.

When enabled, various entries are recorded to `console-services.log` that indicate whether the service is enabled and when each synchronization event has completed. If you enabled SAML after LDAP, these logs can show tokens being revoked in association with past LDAP users if those users haven't logged in through SAML.

Related information

[Require LDAP group membership to log in](#) on page 267

You can use the `exclude-groupless-ldap-users` setting to prevent LDAP users with no group bindings from logging in and creating Puppet Enterprise (PE) accounts. This setting is disabled by default.

[Console and console-services parameters](#) on page 134

In the **PE Console** node group, these parameters customize the behavior of the console and the `console-services` service.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

RBAC database configuration

Credential information for the RBAC service is stored in a PostgreSQL database. Configuration information for this database is in the `rbac-database` section of the config file.

For example:

```
rbac-database: {
  classname: org.postgresql.Driver
  subprotocol: postgresql
  subname: "//<PATH_TO_HOST>:5432/perbac"
  user: <USERNAME>
  password: <PASSWORD>
}
```

It contains these parameters:

classname

Used by the RBAC service for connecting to the database.

The value must be `org.postgresql.Driver`.

subprotocol

Used by the RBAC service for connecting to the database.

The value must always be `postgresql`.

subname

The JDBC connection path used by the RBAC service for connecting to the database.

The value is a string-formatted URI path consisting of the hostname and configured port for the PostgreSQL database along with `perbac`, such as

```
"//<PATH_TO_HOST>:5432/perbac"
```

`perbac` is the database the RBAC service uses to store credentials.

user

This is the username the RBAC service uses to connect to the PostgreSQL database.

password

This is the password the RBAC service uses to connect to the PostgreSQL database.

Configure the password algorithm

Puppet Enterprise (PE) uses SHA-256 as a default password algorithm. You can use Hiera or the PE console to change the algorithm to argon2id by editing or adding password algorithm parameters.

Before you begin: Before changing your password algorithm to argon2id, review the [Argon2 specifications on password-hashing.net](#).

Restriction: If you have [FIPS 140-2 enabled PE](#) on page 17, use the default SHA-256 algorithm, because Argon2id isn't available for FIPS-enabled systems.

puppet_enterprise::profile::console::password_algorithm

A string, either "SHA-256" or "ARGON2ID".

Always required.

Default: "SHA-256"

puppet_enterprise::profile::console::password_hash_output_size

An integer representing the desired hash output size in bytes.

Required for argon2id.

Default: 128 bytes

puppet_enterprise::profile::console::password_algorithm_parallelism

An integer representing the number of parallel computations that can be performed at once.

Required for argon2id.

Default: Twice the number of cores in your system.

puppet_enterprise::profile::console::password_algorithm_memory_in_kb

An integer representing the amount of memory, in KB, the algorithm consumes when running.

Required for argon2id.

No default value. We recommend initially setting this to 25% of your CPU memory.

puppet_enterprise::profile::console::number_of_iterations

An integer representing the number of times a password is hashed before it's stored.

Always required, and we recommend updating this value when switching from SHA-256 to argon2id. The minimum recommended value for argon2id is 3 iterations.

Default: 500000 iterations.

puppet_enterprise::profile::console::password_salt_size_bytes

An integer representing the size, in bytes, of each generated salt.

Default: 128 bytes

Related information

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

Security warnings due to missing HSTS response headers

Puppet Enterprise (PE) does not implement HTTP Strict Transport Security (HSTS) in response headers because the ports used by PE are not open to the internet. However, your security system might flag HSTS response headers as missing and deliver a warning that PE service ports are vulnerable. If this happens, consider adjusting your security software configuration to add an exception for PE ports.

About HSTS

HSTS is designed to protect sites against man-in-the-middle attacks. When HSTS is enabled, an HSTS response header forces user agents and browsers to use HTTPS for loading site content.

Why HSTS is not required for PE

HSTS is not required for PE because none of the ports used by PE are open to the internet.

Adding exceptions for PE ports

If your security system flags a vulnerability due to missing HSTS headers in PE service ports, consider adjusting your security software configuration to add an exception for PE ports.

Typically, an exception is required only for port 443, which is used for PE console services. Port 443 is available to PE users, only within an internal network. To prevent attacks, the console service allows only secure, domain-bound cookies and HTTPS traffic. Mixed content (a combination of HTTP and HTTPS content) is not allowed.

To learn more about PE ports, see [Firewall configuration](#).

Configure proxies

If you have components with limited (or no) internet access, you can configure 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.

Download agent installation packages through a proxy

If your Puppet Enterprise (PE) primary server is airgapped, 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 in the `pe_repo` class.

You must specify `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` in the **PE Master** node group's `pe_repo` class. You can do this in the PE console, the primary server's `pe.conf` file, or Hiera.

To do this in the console, go to **Node Groups > PE Master > Classes**, locate the `pe_repo` class, and set the `pe_repo::http_proxy_host` and `pe_repo::http_proxy_port` parameters.

To do this in the `pe.conf` file, add the following lines to the primary server's `pe.conf` file. Make sure to use values specific to your proxy.

```
"pe_repo::http_proxy_host": "proxy.example.vlan",
"pe_repo::http_proxy_port": 8080
```

Tip: You can use this curl command to test the proxy's connection to the `pe_repo`:

```
proxy_uri='http://<HTTP_PROXY_HOST>:<HTTP_PROXY_PORT>'
uri='https://pm.puppetlabs.com'

curl --proxy "$proxy_uri" --head "$uri"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Related information

[Installing agents](#) on page 145

Puppet Enterprise (PE) agent nodes monitor your infrastructure and help keep it in your desired state. You can install agents on *nix, Windows, and macOS nodes.

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

[How to configure PE](#) on page 211

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. 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.

Set a proxy for agent traffic

General proxy settings in an agent node's `puppet.conf` file are used to manage HTTP connections directly initiated by the agent node.

To configure agents to communicate through a proxy, you must set the `http_proxy_host` and `http_proxy_port` settings in the agent node's `puppet.conf` file.

1. On the agent node, open the `puppet.conf` file, which is located at: `/etc/puppetlabs/puppet/puppet.conf`
2. Add the following lines to the file, with values specific to your proxy:

```
http_proxy_host = proxy.example.vlan
http_proxy_port = 8080
```

For more information about HTTP proxy host options, including `no_proxy`, go to the `http_proxy_host` entry in the Puppet [Configuration Reference](#).

Tip: You can [Configure PXP agent parameters](#) on page 238 to set proxies for PXP agents.

Related information

[Customize the install script](#) on page 148

If necessary, you can use these options to modify the install script to define specific agent configuration settings, CSR attributes, or MSI properties. You can also control whether the Puppet service is running or enabled after agent installation.

Set proxies for Code Manager traffic

Code Manager has proxy configuration options you can use to set proxies for connections to your Git server, the Forge, specific Git repositories, or all Code Manager operations over HTTP(S) transports.

Because Code Manager is run by Puppet Server, Code Manager's proxy settings aren't affected by proxy settings in `puppet.conf` (such as those to [Set a proxy for agent traffic](#) on page 229).

There are several levels and varieties of Code Manager proxy settings. You can:

- Set the `r10k_proxy` parameter in the base [Code Manager settings](#) on page 782, for example:

```
puppet_enterprise::profile::master::r10k_proxy: "http://proxy.example.vlan:8080"
```

Restriction: If you set the `r10k_proxy` parameter, you must use an HTTP URL for the `r10k_remote` parameter and all Puppetfile module entries.

The `r10k_remote` parameter is set when you [Enable Code Manager](#) on page 779. For information about Puppetfile module entries, refer to [Managing modules with a Puppetfile](#) on page 768.

- [Customize Code Manager configuration in Hiera](#) on page 785 to set a global proxy for all HTTP(S) operations, specific proxies for Git and Forge operations, or specific proxies for individual Git repositories.

You can use these settings in combination to override other proxy settings. For example, you can specify a global proxy and a different proxy for Forge operations.

Related information

[Configuring proxies](#) on page 789

If you need Code Manager to use a proxy connection, use the `proxy` parameter. You can set a global proxy for all HTTP(S) operations, proxies for Git or Forge operations, or proxies for individual Git repositories.

Configure the console

After installing Puppet Enterprise (PE), you can change product settings to customize the PE console's behavior. You can configure many of these settings directly in the console.

Configure the PE console and `console-services`

You can configure the behavior of the console and the `console-services` service.

You can set [Password complexity parameters](#) on page 233 and a variety of other [Console and console-services parameters](#) on page 134, such as `rbac_token_maximum_lifetime` or `display_local_time`.

Parameters are set in the PE console, with Hiera, or in `pe.conf`. To configure settings in the PE console:

- Click **Node groups**, and select the node group that contains the class you want to configure.
- On the **Classes** tab, find the class you want to work with, select the **Parameter name** from the list and edit its value.
- Click **Add parameter** and commit changes.
- On the nodes hosting your primary server and PE console, run Puppet.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Related information

[Configuring Puppet orchestrator](#) on page 597

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

[Running Puppet on nodes](#) on page 438

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 trigger a Puppet run.

[Run Puppet on demand](#) on page 604

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Configure settings in pe.conf](#) on page 213

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.

Console and console-services parameters

In the **PE Console** node group, these parameters customize the behavior of the console and the `console-services` service.

You can modify parameters that begin with `puppet_enterprise::profile` in the PE console.

puppet_enterprise::profile::console::classifier_synchronization_period

An integer representing, in seconds, the classifier synchronization period. This controls how long the node classifier can take to retrieve classes from the primary server.

Default: 600

puppet_enterprise::profile::console::ldap_sync_period_seconds

An integer specifying, in seconds, the interval at which LDAP user details and group membership associations are synchronized.

The default value is 1800 (30 minutes).

This synchronization refreshes the user details and group membership for every LDAP user in the system, regardless of the last time the user logged in. If a user is no longer present in LDAP, all user-group associations are removed from the user and all of the user's known tokens are revoked.

To disable automatic synchronization, set the value to 0 or a negative integer. When disabled, user details and group membership only refresh when the user logs in.

When enabled, various entries are recorded to `console-services.log` that indicate whether the service is enabled and when each synchronization event has completed.

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.

Default: 10

puppet_enterprise::profile::console::rbac_password_reset_expiration

An integer representing the number of hours that `password reset` tokens are valid.

An administrator generates these token for users to reset their passwords.

Default: 24

puppet_enterprise::profile::console::rbac_session_timeout

An 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

puppet_enterprise::profile::console::session_maximum_lifetime

A string representing how long a console session can last.

The value is formatted as a string consisting of a number and an optional suffix representing a unit of time: `s` (seconds), `m` (minutes), `h` (hours), `d` (days), or `y` (years).

Example: `"1d"` (one day)

If the suffix is omitted, the default unit of time is seconds.

A value of `"0"` sets an unlimited console session time.

To prevent console sessions from expiring before the maximum RBAC token lifetime, set this parameter to `"0"`.

puppet_enterprise::profile::console::rbac_token_auth_lifetime

A string representing the default authentication lifetime for a token.

The value is formatted as a string consisting of a number followed by a suffix representing a unit of time: **y** (years), **d** (days), **h** (hours), **m** (minutes), or **s** (seconds).

Important: This value cannot exceed the `rbac_token_maximum_lifetime`.

Default: "1h" (one hour)

puppet_enterprise::profile::console::rbac_token_maximum_lifetime

A string representing the maximum allowable lifetime for all tokens.

The value is formatted as a string consisting of a number followed by a suffix representing a unit of time: **y** (years), **d** (days), **h** (hours), **m** (minutes), or **s** (seconds).

Default: 10y (10 years)

puppet_enterprise::profile::console::console_ssl_listen_port

An integer representing the port that the console listens on.

Default: 443

puppet_enterprise::profile::console::ssl_listen_address

A string containing an IP address representing the console's NGINX listen address.

Default: "0.0.0.0"

puppet_enterprise::profile::console::classifier_prune_threshold

An integer representing the number of days to wait before pruning the node classifier database. The node classifier database contains node check-in history if `classifier_node_check_in_storage` is enabled.

Set the value to 0 to never prune the node classifier database.

Default: 7 (days), but only has data to prune if `classifier_node_check_in_storage` is true.

puppet_enterprise::profile::console::classifier_node_check_in_storage

A Boolean specifying whether to create records when nodes check in with the node classifier. These records describe how nodes match the node groups they're classified into.

Set to `true` to enable node check-in storage. Enabling this parameter is required to use [Nodes check-in history endpoints](#) on page 548.

Set to `false` to disable node check-in storage.

Default: `false`

puppet_enterprise::profile::console::display_local_time

A Boolean indicating whether to show timestamps in the local time or UTC.

Set to `true` to display timestamps in local time with hover text showing the equivalent UTC time.

Set to `false` to show timestamps in UTC time with no hover text.

Default: `false`

puppet_enterprise::profile::console::disclaimer_content_path

Specifies the location of the `disclaimer.txt` file containing disclaimer content that can appear on the console login page if you [Create a custom login disclaimer](#) on page 267.

Default: "/etc/puppetlabs/console-services"

Tip: You can also use the [RBAC API Disclaimer endpoints](#) on page 357 to configure the disclaimer without needing to reference a specific file location on disk.

The parameters must be set in Hiera or `pe.conf`, not the console:

puppet_enterprise::api_port

An integer specifying the 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)

For more information, refer to [Node run statuses](#) on page 382.

console_admin_password

The password to log into the console as the admin.

Example: "myconsolepassword"

Default: Specified during installation.

Tip: You can also [Reset the console administrator password](#) on page 266 from the command line.

Related information[Create a custom login disclaimer](#) on page 267

You can add a custom banner to console login page. For example, you can add a disclaimer about authorized or unauthorized use of private information found in the console.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

Password complexity parameters

These parameters set complexity requirements for new passwords created by local users.

Important: Changing password complexity requirements doesn't impact local users' existing passwords.

Requirements are enforced only when creating or changing a password.

puppet_enterprise::profile::console::login_minimum_length

An integer specifying the minimum number of characters required in a login (user name). For example, user names must be at least six characters.

Default: 1

puppet_enterprise::profile::console::password_minimum_length

An integer specifying the minimum number of characters required in a password. For example, passwords must be at least eight characters.

Default: 8

puppet_enterprise::profile::console::letters_required

An integer specifying the minimum number of letter characters required in a password. For example, passwords must have at least one letter.

Default: 0

puppet_enterprise::profile::console::lowercase_letters_required

An integer specifying the minimum number of lowercase letter characters required in a password. For example, passwords must have at least one lowercase letter.

Default: 0

puppet_enterprise::profile::console::uppercase_letters_required

An integer specifying the minimum number of capital letter characters required in a password. For example, passwords must have at least one capital letter.

Default: 0

puppet_enterprise::profile::console::numbers_required

An integer specifying the minimum number of number characters required in a password. For example, passwords must have at least one number, 0 through 9.

Default: 0

puppet_enterprise::profile::console::special_characters_required

An integer specifying the minimum number of special characters required in a password, such as @, #, \$, or !. For example, passwords must have at least one special character.

Default: 0

puppet_enterprise::profile::console::number_of_previous_passwords

An integer specifying the number of previous passwords the system remembers so they can't be reused when a user changes their password. For example, a user's new password can't be the same as any of the user's previous three passwords.

Default: 0

puppet_enterprise::profile::console::username_substring_match

A Boolean specifying whether to compare logins (user names) and passwords for uniqueness.

Set to `true` to apply the `substring_character_limit` and prevent users from creating login-password combinations where the password is completely or partially the same as the login.

Default: `false`

puppet_enterprise::profile::console::substring_character_limit

An integer specifying how many consecutive characters from the login (user name) can appear in the password.

For example, passwords cannot include more than three consecutive characters from the login.

Default: 0

For RBAC-related parameters, such as `rbac_failed_attempts_lockout`, refer to [Console and console-services parameters](#) on page 134 and [Configure RBAC and token-based authentication settings](#) on page 224.

Manage the HTTPS redirect

By default, the Puppet Enterprise (PE) console redirects to HTTPS when you attempt to connect over HTTP. You can customize the redirect target URL or disable redirection.

Set the HTTPS redirect target URL

The default redirect target URL is your primary server's FQDN. You can customize the redirect URL.

To change the redirect target URL:

1. In the PE console, 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.
3. Add the `server_name` parameter and change the value to the desired server.
4. Click **Add parameter** and commit changes.
5. On the nodes hosting your primary server and PE console, run Puppet.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Disable the HTTPS redirect

By default, the `pe-nginx` webserver listens on port 80. If you need to run your own service on port 80, you can use Hiera to disable the HTTPS redirect.

1. Open your default Hiera `.yaml` file in a text editor.

Tip: For information about Hiera data files, including file paths, refer to [Configure settings with Hiera](#) on page 212.

2. Add the `enable_http_redirect` parameter and set to `false`. For example:

```
puppet_enterprise::profile::console::proxy::http_redirect::enable_http_redirect:
  false
```

3. Save the file and run `puppet agent -t` to compile the changes.

Enable data editing in the console

In new Puppet Enterprise (PE) installations, you can, by default, edit configuration data in the console. If you upgraded from an earlier PE version where you hadn't already enabled editing of configuration data, you must use Hiera to manually enable Classifier Configuration Data.

1. On your primary server, open the `hiera.yaml` file located at: `/etc/puppetlabs/puppet/hiera.yaml`.
2. Add the following to the `hiera.yaml` file:

```
hierarchy:
  - name: "Classifier Configuration Data"
    data_hash: classifier_data
```

Place additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same `hierarchy` key, below the `Classifier Configuration Data` entry.

3. To allow users to edit the configuration data in the console, add the **Set environment** and **Edit configuration data** permissions to any user groups that need to set environment parameters or modify class parameters.
4. If your environment is configured for disaster recovery or has compilers, update `hiera.yaml` on your replica and compilers, respectively.

Add custom PQL queries to the console

Add your own Puppet Query Language (PQL) queries to the console to quickly access them when running jobs.

For help forming queries, go to the PQL [Reference guide](#) in the Puppet documentation.

1. On the primary server, copy the `custom_pql_queries.json.example` file, and remove the `.example` suffix. For example, you can use this command:

```
sudo cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

2. Edit the file contents to include your own PQL queries or remove any existing queries.
3. Refresh the console UI in your browser.

You can now see your custom queries in the PQL drop-down options when running jobs.

Configure orchestration

After installing PE, you can change some default settings to further configure the orchestrator and `pe-orchestration-services`.

Related information

[Configuring Puppet orchestrator](#) on page 597

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.

[How Puppet orchestrator works](#) on page 586

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

Orchestrator and pe-orchestration-services parameters

These are some optional parameters you can use to configure the behavior of the orchestrator and the `pe-orchestration-services` service.

You can modify these profile class parameters in the Puppet Enterprise (PE) console on the **Classes** tab for the **PE Orchestrator** infrastructure node group.

`puppet_enterprise::profile::agent::pxp_enabled`

Boolean used to enable or disable the Puppet Execution Protocol (PXP) service.

Set to `true` to enable the PXP service, which is required to use the orchestrator and run Puppet from the console.

Set to `false` to disable the PXP service. If `false`, you can't use the orchestrator or the **Run Puppet** button in the console.

Must be `true` to [Configure PXP agent parameters](#) on page 238.

Default: `true`

`puppet_enterprise::profile::bolt_server::concurrency`

An integer that determines the maximum number of simultaneous task or plan requests the orchestrator can make to `bolt-server`.

This setting only limits task or plan executions on nodes with SSH or WinRM transport methods, because these are the only tasks and plans requiring 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::orchestrator::allowed_pcp_status_requests`

An integer that defines how many times an orchestrator job allows status requests to time out before a job is considered failed. Status requests wait 12 seconds between timeouts, so multiply the value of the `allowed_pcp_status_requests` by 12 to determine how many seconds the orchestrator waits on targets that aren't responding to status requests.

Default: 35 timeouts

`puppet_enterprise::profile::orchestrator::global_concurrent_compiles`

An integer specifying how many concurrent compile requests can be outstanding to the primary server across all orchestrator jobs.

Default: 8 requests

`puppet_enterprise::profile::orchestrator::java_args`

Specifies the [Java heap](#) on page 207 size, which is the amount of JVM memory that each Java process is allowed to request from the OS for orchestration services to use.

The value is formatted as a JSON hash, where the maximum and minimum are usually the same. For example:
`{ "Xmx" : "256m", "Xms" : "256m" }`

Default: 704 MB

`puppet_enterprise::profile::orchestrator::job_prune_threshold`

An integer of 2 or greater, which specifies the number of days to retain job reports.

This parameter sets the corresponding parameter `job-prune-days-threshold`.

While `job_prune_threshold` itself has no default value, `job-prune-days-threshold` has a default of 30 (30 days).

`puppet_enterprise::profile::orchestrator::jruby_max_active_instances`

An integer that determines the maximum number of JRuby instances that the orchestrator creates to execute plans. Because each plan uses one JRuby to run, this value is effectively the maximum number of concurrent plans. Setting the orchestrator heap size (`java_args`) automatically sets the `jruby_max_active_instances` using the formula $\$java_args \div 1024$. If the result is less than one, the default is one JRuby instance.

Default: 1 instance

Note: The `jruby_max_active_instances` pool for the orchestrator is separate from the Puppet Server pool. Refer to [JRuby max active instances](#) on page 206 for more information.

`puppet_enterprise::profile::orchestrator::max_connections_per_route_authenticated`

An integer representing the maximum number of concurrent HTTP-client connections allowed for each route when requests include a client certificate.

Default: 20

`puppet_enterprise::profile::orchestrator::max_connections_per_route_unauthenticated`

An integer representing the maximum number of concurrent HTTP-client connections allowed for each route when requests do not include a client certificate.

Default: 20

`puppet_enterprise::profile::orchestrator::max_connections_total_authenticated`

An integer representing the maximum number of concurrent HTTP-client connections allowed for all routes when requests include a client certificate.

Default: 20

`puppet_enterprise::profile::orchestrator::max_connections_total_unauthenticated`

An integer representing the maximum number of concurrent HTTP-client connections allowed for all routes when requests do not include a client certificate.

Default: 20

`puppet_enterprise::profile::orchestrator::pcp_timeout`

An integer representing how long, in seconds, an agent can spend attempting to connect to a PCP broker during a Puppet run triggered by the orchestrator. If the agent can't connect to the broker in the specified time frame, the Puppet run times out.

Default: 30

`puppet_enterprise::profile::orchestrator::run_service`

A Boolean used to enable (`true`) or disable (`false`) orchestration services.

Default: `true`

`puppet_enterprise::profile::orchestrator::socket_timeout`

An integer specifying, in milliseconds, the maximum wait time before the orchestrator closes an HTTP connection when no data is available on the socket.

Default: 120000

`puppet_enterprise::profile::orchestrator::task_concurrency`

An integer representing the number of simultaneous task or plan actions that can run at the same time. All task and plan actions are limited by this concurrency limit regardless of transport type (WinRM, SSH, PCP).

If a task or plan action runs on multiple nodes, each node consumes one action. For example, if a task needs to run on 300 nodes, and your `task_concurrency` is set to 200, then the task can run on 200 nodes while the remaining 100 nodes wait in queue.

Default: 250 actions

`puppet_enterprise::profile::plan_executor::versioned_deploys`

A Boolean used for [Running plans alongside code deployments](#) on page 651.

Set to `true` to enable versioned deployments of environment code.

Default: `false`

Important: Setting this to `true` **disables** the file sync client's locking mechanism that usually enforces a consistent environment state for your plans. This can cause Puppet functions and plans that call other plans to behave unexpectedly if a code deployment occurs while a plan is running.

For information about how the orchestrator works, what you can do with it, and additional parameters and configuration options, refer to [Orchestrating Puppet runs, tasks, and plans](#) on page 585.

For PXP agent parameters, refer to [Configure PXP agent parameters](#) on page 238.

Related information

[Configuring Puppet orchestrator](#) on page 597

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

[Running Puppet on nodes](#) on page 438

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 trigger a Puppet run.

[Run Puppet on demand](#) on page 604

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

Configure PXP agent parameters

Puppet Execution Protocol (PXP) is a message format used to request task execution and receive task statuses. PXP agents runs the PXP service. You can configure `pxp_agent` parameters with Hiera or in the PE console.

`puppet_enterprise::profile::agent::pxp_enabled`

Boolean used to enable or disable the Puppet Execution Protocol (PXP) service.

Set to `true` to enable the PXP service, which is required to use the orchestrator and run Puppet from the console.

Set to `false` to disable the PXP service. If `false`, you can't use the orchestrator or the **Run Puppet** button in the console.

Default: `true`

`puppet_enterprise::pxp_agent::ping_interval`

An integer specifying the frequency, in seconds, that PXP agents ping PCP brokers. If the broker doesn't respond, the agent tries to reconnect.

Default: 120

`puppet_enterprise::pxp_agent::pxp_logfile`

The path, as a string, to the PXP agent log file. This file can be used to debug orchestrator issues.

The default value varies by OS.

- *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

A string representing the amount of time to retain records of Puppet or task runs on agents.

Format the value as a string consisting of a number and one of the following suffixes: m (minutes), h (hours), or d (days).

Default: "14d" (14 days)

puppet_enterprise::pxp_agent::task_cache_dir_purge_ttl

A string representing the amount of time that task files are cached after use.

Format the value as a string consisting of a number and one of the following suffixes: m (minutes), h (hours), or d (days).

Default: "14d" (14 days)

puppet_enterprise::pxp_agent::broker_proxy

Optional. Set a proxy URI to use to connect to the pcp-broker to listen for task and Puppet run requests.

puppet_enterprise::pxp_agent::master_proxy

Optional. Set a proxy URI to use to connect to the primary server to download task implementations.

puppet_enterprise::pcp_max_message_size_mb

An integer specifying the maximum message size, in MB, for pcp_broker, ppxp_agent, and the orchestrator.

The maximum message size cannot be higher than the default size of 64 MB. You can only specify a smaller value.

Default: 64

Important: Don't change the pcp_max_message_size_mb parameter if you send or receive large payloads, because this can cause errors for large task and plan run parameters and output.

Related information

[Puppet orchestrator technical overview](#) on page 586

The orchestrator uses `pe-orchestration-services`, a JVM-based service in Puppet Enterprise (PE), to execute on-demand Puppet runs on agent nodes in your infrastructure. The orchestrator uses Puppet Execution Protocol (PXP) agents to orchestrate changes across your infrastructure.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

Manage ARP table overflow

In larger deployments that use the PCP broker, you might encounter Address Resolution Protocol (ARP) table overflows.

Overflows occur when the ARP table (which is a local cache of IP-to-MAC-address resolutions) becomes full and starts evicting old entries. When long-established, but frequently-used, entries are evicted, network traffic increases to restore them. This increases network latency and CPU load on the broker.

Here is an example of a typical ARP table overflow log message:

```
[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 resolve this issue, you need to increase `sysctl` settings related to ARP tables.

For example, these 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
```

Configure ulimit

As your infrastructure grows and you use Puppet Enterprise (PE) to manage more agents, you might need to increase the number of allowed file handles per client.

PE services can require as much as one file handle per connected client. The default ulimit settings for most operating systems can only support up to about 200 clients. To support more clients, you need to increase the number of allowed file handles.

You can increase file handle limits for these PE services:

- `pe-orchestration-services`
- `pe-puppetdb`
- `pe-console-services`
- `pe-puppetserver`
- `pe-puppet`

Where and how you configure ulimit depends on the agent's platform. We've provided instructions to:

- [Configure ulimit using systemd](#) on page 240
- [Configure ulimit using upstart](#) on page 241
- [Configure ulimit on other init systems](#) on page 241

Tip: In these instructions, replace `<PE_SERVICE>` with the name of the service you're configuring. For example, if you're configuring ulimit for the PuppetDB service, replace `<PE_SERVICE>` with `pe-puppetdb`.

Additionally, these instructions use 32678 as a sample ulimit value. Change this value according to your needs.

Configure ulimit using systemd

With systemd, the allowed number of open file handles is controlled by the `LimitNOFILE` setting in the `.service` file each PE service.

1. Locate the systemd `.service` file for the PE service you want to configure and copy the file path. The default file path is:

```
/usr/lib/systemd/system/<PE_SERVICE>.service
```

For example, the file path for the PuppetDB service systemd file is:

```
/usr/lib/systemd/system/pe-puppetdb.service
```

For a list of service names, refer to [Configure ulimit](#) on page 240.

- Using the file path you determined in the previous step, run the following commands to increase the ulimit. Make sure to set the `LimitNOFILE` value to the desired file handles limit.

```
mkdir /etc/systemd/system/<PE_SERVICE>.service.d
echo "[Service]LimitNOFILE=32678" > /etc/systemd/system/
<PE_SERVICE>.service.d/limits.conf
systemctl daemon-reload
```

- To confirm the change, run:

```
systemctl show <PE_SERVICE> | grep LimitNOFILE
```

- Repeat these steps to configure ulimit for other PE services.

Configure ulimit using upstart

For Ubuntu and Red Hat systems, the allowed number of open file handles is specified in system configuration files for each PE service.

- Locate the file for the PE service you want to configure. The location depends on the platform, and the file name matches the PE service name (as listed in [Configure ulimit](#) on page 240).
 - Ubuntu: `/etc/default/<PE_SERVICE>`
 - Red Hat: `/etc/sysconfig/<PE_SERVICE>`
- Set the `ulimit` setting on the last line of the file as follows:

```
ulimit -n <ULIMIT_VALUE>
```

For example, this configuration set the allowed number of open files to 32,678:

```
ulimit -n 32678
```

Configure ulimit on other init systems

The `ulimit` controls the number of processes and file handles that a PE service user can open and process.

To increase the ulimit for a PE service user:

- Open the `limits.conf` file located at: `/etc/security/limits.conf`
- Add these lines to the file, specifying the specific service user's name and the desired ulimit value:

```
<PE_SERVICE_USER> soft nofile <VALUE>
<PE_SERVICE_USER> hard nofile <VALUE>
```

For example, this configuration sets the ulimit value to 32,678 for the `pe-puppet` service user:

```
pe-puppet soft nofile 32678
pe-puppet hard nofile 32678
```

Analytics data collection

Some components automatically collect data about how you use Puppet Enterprise (PE). You can opt out of this data collection during or after installing PE.

This data is separate from other analytics data, such as that collected by the [PE support script](#) on page 22, [Puppet metrics collector](#) on page 21, the [pe_status_check module](#), the [Value report](#) on page 389, or the analytics trapperkeeper service (which is described in [Orchestration services settings](#) on page 597).

How does sharing this data benefit you?

We use this data to identify customers that could be impacted by security issues, alert them to the issue, and provide them with relevant instructions to follow or fixes to download.

Additionally, the data helps us understand how customers use PE. This helps us improve PE in ways that benefit you.

How does Puppet by Perforce use the collected data?

This data collection is one of many ways we learn about our customers. For example, knowing how many nodes you manage helps us develop more realistic product testing. Similarly, learning which operating systems are the most and the least popular helps us prioritize development. By sharing this data, we can better understand your experience as a PE customer.

Important: Collected data is tied to a unique, anonymized identifier for each primary server and your site as a whole. We don't collect any personally identifiable information (PII), and we never use or share collected data outside Puppet by Perforce.

What data does PE collect?

Puppet Enterprise (PE) collects the following data when Puppet Server starts, restarts, and every 24 hours of continuous runtime.

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 on 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

PE feature use information

- Number of node groups in use
- Number of nodes used in orchestrator jobs after the last orchestrator restart
- Mean number of nodes per orchestrator job
- Maximum number of nodes per orchestrator job
- Minimum number of nodes per orchestrator job
- Total number of orchestrator jobs created after the 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 (PCP)
- 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 that were 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 (used to distinguish between jobs started via the console, command line, or API)
- UUID used to correlate multiple jobs run by the same user
- Time at which the task job ran
- 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 ran
- Type of version control system webhook
- Whether the request was to deploy all environments
- Whether Code Manager waited for all deploys to finish (successfully or with errors) before returning a response
- Whether the deploy was a dry run
- List of environments requested to deploy
- List of deploy requests
- Total time elapsed for all deploys to finish (successfully or with errors)
- List of total wait times for deploys specifying the `--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 the user used the `--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 the user used the `--force` option when running `puppet-backup create`
- Whether the user used the `--dir` option when running `puppet-backup create`
- Whether the user used the `--name` option when running `puppet-backup create`
- Scope of backup
- Time in seconds for various backup functions
- Time needed to estimate backup size, disk space needed, and disk space available
- Time to create file system backup
- Time to back up the `pe-orchestrator` database
- Time to back up the `pe-rbac` database
- Time to back up the `pe-classifier` database
- Time to back up the `pe-activity` database
- Time to back up the `pe-puppetdb` database
- Time to compress archive file to backup directory
- Time to back up PE-related classification
- Total time to back up
- List of `puppet-backup create` jobs

Puppet Server performance information

- Total number of JRuby instances
- Maximum number of active JRuby instances
- Maximum number of requests per JRuby instance
- Average number of instances not used over the process' 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, or repair)
- Current version, if upgrading
- Target version
- Whether installation succeeded or failed, and limited failure type information (if applicable)

If you use an Amazon Web Services Marketplace Image to install PE, this information is also collected:

- Marketplace name
- Marketplace image billing mode (*bring your own license* or *pay as you go*)

PE console information

While in use, the PE console collects:

- Pageviews
- Link and button clicks
- Page load time
- User language
- Screen resolution
- Viewport size
- Anonymized IP address

Important: The console **doesn't** collect user inputs, such as node or group names, user names, rules, parameters, or variables.

Opt out when installing PE

You can set the `DISABLE_ANALYTICS` environment variable when you run the install script.

When you [Install PE using the installer tarball](#) on page 125, add `DISABLE_ANALYTICS=1` when you call the installer script, for example:

```
sudo DISABLE_ANALYTICS=1 ./puppet-enterprise-installer
```

Including this option with the install script command sets the `puppet_enterprise::send_analytics_data` parameter to `false` in the `pe.conf` file, which disables collection of the data described in [What data does PE collect?](#) on page 242.

Opt out after installing PE

You can opt out of analytics data collection after installing PE. You can also use these steps to enable data collection if you want to opt in.

1. In the PE console, go to **Node groups > PE Infrastructure**.
 2. On the **PE Infrastructure** page, select the **Classes** tab and locate the `puppet_enterprise` class.
 3. From the **Parameter name** drop-down list, select `send_analytics_data` and enter `false` in the **Value** field.
- If you want to opt in to data collection after previously opting out, set the **Value** to `true`.
4. Click **Add to node group** and commit your change by clicking the button in the lower right corner of the page.
 5. Go to the **Nodes** page and select your primary server.
 6. Click the **Run** button and select **Puppet**.

You are taken automatically to the **Jobs** page, where you can run Puppet to enforce the change on the nodes hosting your primary server and PE console.

Tip: There are several ways to target nodes for your jobs. For more information, see [Run Puppet on demand from the console](#) on page 604.

After the Puppet runs complete, the data described in [What data does PE collect?](#) on page 242 is no longer collected (if you set the value to `false`).

Static catalogs

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. Puppet Enterprise (PE) primary servers typically compile catalogs from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata specifying 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 a file (other than the latest version), and it can confirm that the agent is applying the desired version of the file resource for the catalog. Including this metadata in the catalog reduces the number of requests agents make to the primary server.

Go to the Puppet documentation for details about [Resources](#), [File types](#), and [Catalog compilation](#).

About static catalogs

When a primary server produces a non-static catalog, the catalog doesn't specify file resource versions. When an agent applies a non-static catalog, it retrieves the latest version of each file resource or uses a previously-retrieved version (if it matches the latest version's contents). Enable static catalogs if you don't always want to apply the latest version of a file resource.

When a manifest depends on a file with content that changes more frequently than the agent receives new catalogs, a node might apply a version of the referenced file that doesn't match the instructions in the catalog. Consequently, the agent's Puppet runs might produce different results each time the agent applies the same catalog. This often causes problems because Puppet generally expects a catalog to produce the same results each time it's applied, regardless of any code or file content updates on the primary server.

Additionally, each time an agent applies a normal cached catalog that contains file resources sourced from `puppet:///` locations on the node, the agent requests file metadata from the primary server each time it applies the catalog, even if nothing has changed in the cached catalog. This causes the primary server to perform unnecessary resource-intensive checksum calculations for each file resource.

Note: These potential issues only impact file resources that use the `source` attribute. File resources that use the `content` attribute aren't impacted, and their behavior does not change in static catalogs.

Static catalogs avoid these problems by including metadata referencing specific versions of file resources. This prevents newer versions from being applied incorrectly, and it prevents forcing agents to regenerate metadata on each Puppet run. The metadata is delivered in the form of a unique hash maintained, by default, by the file sync service.

The catalog is referred to as *static* because it contains all the information that an agent needs to determine whether the node's configuration matches instructions (for using file resources) and the state of file resources at the static point in time when the catalog was generated.

Differences in catalog behavior

Without static catalogs enabled:

1. The agent 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 described in the catalog. If the agent finds any resources that are not in the desired state, it makes the necessary changes.

With static catalogs enabled:

1. The agent sends facts to the primary server and requests a catalog.
2. The primary server compiles and returns the agent's catalog, including metadata specifying the desired state of the node's file resources.
3. The agent applies the catalog by checking each resource described in the catalog. 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.
4. 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, which requires [Managing code with Code Manager](#) on page 774.

After enabling file sync (by enabling Code Manager), Puppet Server automatically creates static catalogs containing file metadata for eligible resources, and agents running Puppet 1.4.0 or newer can leverage static catalog features.

It is possible to use static catalogs without file sync or Code Manager, but you'll need to create custom scripts and set certain 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.

Static catalogs exceptions

There are some scenarios when the benefits of static catalogs aren't realized. Agents always process catalogs in these scenarios, but without the benefits of in-lined file metadata or file resource versions.

In these scenarios static catalogs either aren't applied by agents or don't include metadata for file resources:

- Static catalogs are disabled globally.
- Code Manager and file sync are disabled.
- If you're using static catalogs without file sync and you haven't configured the `code_id` and `code_content` parameters.
- Your agents aren't running PE 2016.1 or later (which includes Puppet agent version 1.4.0 or later).

Catalogs don't include metadata for file resources if the file resource:

- Uses the `content` attribute instead of the `source` attribute.
- Uses the `source` attribute with a non-Puppet scheme, such as:

```
source => 'http://host:port/path/to/file'
```

- Uses the `source` attribute without the built-in modules mount point.
- Uses the `source` attribute, but, on the primary server, the file is not located at:

```
/etc/puppetlabs/code/environments/<environment>/**/files/**
```

For example, module files are typically located at:

```
/etc/puppetlabs/code/environments/<environment>/modules/<module_name>/
files/**
```

Related information

[Enabling or disabling file sync](#) on page 819

File sync is normally enabled or disabled automatically along with Code Manager.

[How Puppet orchestrator works](#) on page 586

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

Globally disable static catalogs

You can use Hiera to prevent use of static catalogs across your infrastructure.

1. Open your default Hiera `.yaml` file in a text editor.

Tip: For information about Hiera data files, including file paths, refer to [Configure settings with Hiera](#) on page 212.

2. Add the `static_catalogs` parameter and set the value to `false`. For example:

```
puppet_enterprise::master::static_catalogs: false
```

3. Save the file and run `puppet agent -t` to compile the changes.

Use static catalogs without file sync

To use static catalogs without enabling file sync or Code Manager, you must set the `code_id` and `code_content` parameters, and then configure the `code_id_command`, `code_content_command`, and `file_sync_enabled` parameters in the Puppet Enterprise (PE) console.

1. Follow the instructions for [Configuring code_id and the static_file_content endpoint](#) in the open-source Puppet documentation to set the `code_id` and `code_content` settings.
2. In the PE console, go to **Node groups > PE Infrastructure > PE Master**.
3. On the **Classes** tab, locate the `puppet_enterprise::profile::master` class.
4. Add the `file_sync_enabled` parameter, set the **Value** to `false`, and click **Add parameter**.
5. Add the `code_id_command` parameter, set the **Value** to the absolute path to the `code_id` script, and click **Add parameter**.
6. Add the `code_content_command` parameter, set the **Value** to the absolute path to the `code_content` script, and click **Add parameter**.
7. Commit changes.
8. Run Puppet on your primary server.

Related information

[Configuring Puppet orchestrator](#) on page 597

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

[Running Puppet on nodes](#) on page 438

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 trigger a Puppet run.

[Run Puppet on demand](#) on page 604

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

[Configure settings in the PE console](#) on page 211

You can use the Puppet Enterprise (PE) console's graphical interface to configure settings for your PE installation.

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 248

Disaster recovery creates a replica of your primary server.

- [Configure disaster recovery](#) on page 258

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.

There are two main advantages to enabling disaster recovery:

- If your primary server fails, the replica takes over the handling of Puppet Server and PuppetDB traffic, allowing existing agents to remain operational and Puppet runs to continue without interruption. By configuring nodes

to automatically fail over to the replica when the primary is unreachable, you can ensure that they still receive catalogs and enforce your desired state.

- 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.

Disaster recovery architecture

The replica is not an exact copy of the primary server. Rather, the replica duplicates specific infrastructure components and services. By default Hiera data and other custom configurations are not replicated. However, if you store Hiera data in the control repository, as recommended, the data is replicated through Code Manager.

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.

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
Orchestration service	read-only	promoted
Console service UI	none	promoted
Agentless Catalog Executor (ACE) service	none	promoted
Bolt service	none	promoted
Host Action Collector service	read-only	promoted

The following services performed by the primary server are unavailable on a replica until the replica is promoted:

- **Certificate authority:** The replica cannot provision new agents.
- **Orchestration:** Tasks, plans, and Puppet runs can not be initiated from the replica. This includes running operations via the Agentless Catalog Executor.
- **Console:** The console is not available on the replica, and classification changes cannot be made from the replica.

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 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (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
- Running Puppet tasks or plans through the orchestrator.

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.
Software	<p>• You must use Code Manager so that code is deployed to both the primary server and the replica after you enable a replica. Code Manager also replicates the certificate authority state, as well as PE configuration files. Even if you have an alternate method for syncing your code across nodes, Code Manager must still be enabled.</p> <p>• You must use the default PE node classifier so that disaster recovery classification can be applied to nodes.</p> <p>• Orchestrator must be enabled so that it can perform PE maintenance and upgrade actions.</p>

Component	Requirement
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. Must use the same operating system type and version as your primary server. Must have the same agent version as your primary server.
Firewall	<p>Your replica must comply with the same port requirements as your primary server to ensure that the replica can operate as the primary server during failover. For details, see the firewall configuration requirements for your installation type.</p>
Node names	<p>You must use resolvable domain names when specifying node names for the primary server and replica.</p>
RBAC tokens	<p>You must have an admin RBAC token when running some <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. However, an RBAC token isn't required to promote a replica or to run the <code>enable_ha_failover</code> command.</p>

Related information

[Firewall configuration](#) on page 106

Follow these guidelines for firewall configuration based on your installation type.

[Generate a token using `puppet-access`](#) on page 305

Use the `puppet-access` command to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

Classification changes in disaster recovery installations

When you provision and enable a replica, the system makes a number of classification changes in order to manage disaster recovery.

Two infrastructure node groups are added in installations with disaster recovery. The **PE HA Master** node group includes your primary server and inherits from the **PE Master** node group. The **PE HA Replica** node group includes your replica and inherits from the **PE Infrastructure** node group.

Additional disaster recovery configuration is managed with these parameters:

- `classifier_client_certname`
- `classifier_host`
- `classifier_port`
- `ha_enabled_replicas`
- `manage_puppet_conf`
- `pcp_broker_list`
- `primary_uris`
- `provisioned_replicas`
- `puppetdb_host`
- `puppetdb_port`
- `replica_hostnames`
- `replicating`
- `replication_mode`
- `server_list`
- `sync_allowlist`
- `sync_peers`

Note: Apart from the parameters in the **PE Agent** and **PE Infrastructure Agent** node groups (`manage_puppet_conf`, `server_list`, `pcp_broker_list`, and `primary_uris`), all of these are system parameters that should not be manually modified. The **PE Agent** and **PE Infrastructure Agent** parameters are automatically updated based on the values you specify when you provision and enable a replica.

`classifier_client_certname`

Purpose

Specifies the name on the certificate used by the classifier.

Node group

PE Master

Class

```
puppet_enterprise::profile::master
```

DR-only parameter

No

Example with enabled replica

```
[ "<PRIMARY_CERTNAME>" , "<REPLICA_CERTNAME>" ]
```

Notes

Replica values are appended to the end of parameter when a replica is enabled.

`classifier_host`

Purpose

Specifies the certname of the node running the classifier service.

Node group

PE Master

Class

```
puppet_enterprise::profile::master
```

DR-only parameter

No

Example with enabled replica

```
[ "<PRIMARY_CERTNAME>" , "<REPLICA_CERTNAME>" ]
```

Notes

Replica values are appended to the end of parameter when a replica is enabled.

classifier_port**Purpose**

Specifies the port used for communicating with the classifier service. Always 4433.

Node group**PE Master****Class**

```
puppet_enterprise::profile::master
```

DR-only parameter

No

Example with enabled replica

```
[ 4433 , 4433 ]
```

Notes

Replica values are appended to the end of parameter when a replica is enabled.

ha_enabled_replicas**Purpose**

Tracks replica nodes that are failover ready.

Node group**PE Infrastructure****Class**

```
puppet_enterprise
```

DR-only parameter

Yes

Example with enabled replica

```
[ "<REPLICA_CERTNAME>" ]
```

Notes

Updated when you enable a replica.

manage_puppet_conf**Purpose**

When true, specifies that the server_list setting is managed in puppet.conf.

Node group**PE Agent, PE Infrastructure Agent****Class**

```
puppet_enterprise::profile::agent
```

DR-only parameter

No

Example with enabled replica

```
true
```

pcp_broker_list**Purpose**

Specifies the list of Puppet Communications Protocol brokers that Puppet Execution Protocol agents contact, in order.

Node group

PE Agent, PE Infrastructure Agent

Class

```
puppet_enterprise::profile::agent
```

DR-only parameter

No

Example with enabled replica

PE Agent — ["<PRIMARY_CERTNAME>:8142 , "<REPLICA_CERTNAME>:8142 "] or in a large installation, ["<LOAD_BALANCER>:8142 "]

PE Infrastructure Agent — ["<PRIMARY_CERTNAME>:8142" , "<REPLICA_CERTNAME>:8142"]

Notes

- Infrastructure nodes must be configured to communicate directly with the primary in the **PE Infrastructure Agent** node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the **PE Agent** node group.
- When a replica is enabled, the replica is appended to the end of the list in the **PE Infrastructure Agent** group, and when not using a load balancer, it's appended to the list in **PE Agent**.
- Some `puppet infrastructure` commands refer to this parameter as `agent-server-urls`, but those commands nonetheless manage the `server_list` parameter.

Important: Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported.

primary_uris**Purpose**

Specifies the list of Puppet Server nodes hosting task files for download that Puppet Execution Protocol agents contact, in order.

Node group

PE Agent, PE Infrastructure Agent

Class

```
puppet_enterprise::profile::agent
```

DR-only parameter

No

Example with enabled replica

PE Agent — ["<PRIMARY_CERTNAME>:8140 , "<REPLICA_CERTNAME>:8140 "], or in a large installation, ["<LOAD_BALANCER>:8140 "]

PE Infrastructure Agent — ["<PRIMARY_CERTNAME>:8140" , "<REPLICA_CERTNAME>:8140"]

Notes

- Infrastructure nodes must be configured to communicate directly with the primary in the **PE Infrastructure Agent** node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the **PE Agent** node group.
- When a replica is enabled, the replica is appended to the end of the list in the **PE Infrastructure Agent** group, and when not using a load balancer, it's appended to the list in **PE Agent**.
- Some `puppet infrastructure` commands refer to this parameter as `agent-server-urls`, but those commands nonetheless manage the `server_list` parameter.

Important: Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported.

`provisioned_replicas`

Purpose

Specifies the certname of replica to give access to the ca-data file sync repo.

Node group

PE HA Master

Class

```
puppet_enterprise::profile::master
```

DR-only parameter

Yes

Example with enabled replica

```
[ "<REPLICA_CERTNAME>" ]
```

`puppetdb_host`

Purpose

Specifies the certname of the node running the PuppetDB service.

Node group

PE Master

Class

```
puppet_enterprise::profile::master
```

DR-only parameter

No

Example with enabled replica

```
[ "<PRIMARY_CERTNAME>" , "<REPLICA_CERTNAME>" ]
```

Notes

Replica values are appended to the end of parameter when a replica is enabled.

`puppetdb_port`

Purpose

Specifies the port used for communicating with the PuppetDB service. Always 8081.

Node group

PE Master

Class

```
puppet_enterprise::profile::master
```

DR-only parameter

No

Example with enabled replica

```
[8081,8081]
```

Notes

Replica values are appended to the end of parameter when a replica is enabled.

replica_hostnames**Purpose**

Specifies the certname of the replica to set up pglogical replication for non-PuppetDB databases.

Node group

PE HA Master

Class

```
puppet_enterprise::profile::database
```

DR-only parameter

Yes

Example with enabled replica

```
[ "<REPLICA_CERTNAME>" ]
```

replicating**Purpose**

Specifies whether databases other than PuppetDB replicate data.

Node group

PE Infrastructure

Class

```
puppet_enterprise
```

DR-only parameter

Yes

Example with enabled replica

```
true
```

Notes

Used when provisioning a new replica.

replication_mode**Purpose**

Sets replication type and direction on primary servers and replicas.

Node group

PE Master (none), HA Master (source)

Class

```
puppet_enterprise::profile::master
```

```
puppet_enterprise::profile::database
```

```
puppet_enterprise::profile::console
```

DR-only parameter

Yes (although "none" by default)

Example with enabled replica

PE Master — "none" (Present only in master profile.)

PE HA Master — "source" (Set automatically in the replica profile; no setting in the classifier in **PE HA Replica**.)

server_list

Purpose

Specifies the list of servers that agents contact, in order.

Node group

PE Agent, PE Infrastructure Agent

Class

```
puppet_enterprise::profile::agent
```

DR-only parameter

No

Example with enabled replica

PE Agent — ["<PRIMARY_CERTNAME>:8140", "<REPLICA_CERTNAME>:8140"] or in a large installation, ["<LOAD_BALANCER>:8140"]

PE Infrastructure Agent — ["<primary certname>:8140", "<replica certname>:8140"]

Notes

- Infrastructure nodes must be configured to communicate directly with the primary in the **PE Infrastructure Agent** node group, or in a DR configuration, the primary and then the replica. In large installations with compilers, agents must be configured to communicate with the load balancers or compilers in the **PE Agent** node group.
- When a replica is enabled, the replica is appended to the end of the list in the **PE Infrastructure Agent** group, and when not using a load balancer, it's appended to the list in **PE Agent**.
- Some `puppet infrastructure` commands refer to this parameter as `agent-server-urls`, but those commands nonetheless manage the `server_list` parameter.

Important: Setting agents to communicate directly with the replica in order to use the replica as a compiler is not supported.

sync_allowlist

Purpose

Specifies a list of nodes that the primary PuppetDB syncs with.

Node group

PE HA Master

Class

```
puppet_enterprise::profile::puppetdb
```

DR-only parameter

Yes

Example with enabled replica

["<REPLICA_CERTNAME>"]

During upgrade, when primary is upgraded but replica hasn't been upgraded, [] to prevent syncing until upgrade is complete.

sync_peers

Purpose

Specifies a list of hashes that contain configuration data for syncing with a remote PuppetDB node. Includes the host, port, and sync interval.

Node group

PE HA Master

Class

```
puppet_enterprise::profile::puppetdb
```

DR-only parameter

Yes

Example with enabled replica

```
[ { "host" : "<REPLICA_CERTNAME>" , "port" : 8081 , "sync_interval_minutes" : <X> } ]
```

During upgrade, when primary is upgraded but replica hasn't been upgraded, [] to prevent syncing until upgrade is complete.

Notes

Updated when you enable a replica.

Load balancer timeout in disaster recovery installations

Disaster recovery configuration uses timeouts to determine when to fail over to the replica. If the load balancer timeout is shorter than the server and agent timeout, connections from agents might be terminated during failover.

To avoid timeouts, set the timeout option for load balancers to four minutes or longer. This duration allows compilers enough time for required queries to PuppetDB and the node classifier service. You can set the load balancer timeout option using parameters in the haproxy or f5 modules.

Configure disaster recovery

To configure disaster recovery, you must provision a replica to serve as backup during failovers. If your primary server is permanently disabled, you can then promote a replica.

Before you begin

- Apply [disaster recovery system and software requirements](#).
- Ensure you have [a valid admin RBAC token](#) that is valid for at least an hour, so that it does not expire during the provisioning process. You can delete the token after provisioning is complete.
- 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.
- If you're using an r10k private key for code management, set `puppet_enterprise::profile::master::r10k_private_key` in `pe.conf`. This ensures that the r10k private key is synced to your primary server replica.
- [Back up your classifier hierarchy](#), because enabling a replica alters classification.

Tip: Some of the `puppet infrastructure` commands that are used to configure and manage disaster recovery require a valid admin RBAC token, and all commands must be run from a root session. Running with elevated privileges via `sudo puppet infrastructure` is not sufficient. Instead, start a root session by running `sudo`

`su -`, and then run the `puppet infrastructure` command. For details about these commands, run `puppet infrastructure help <ACTION>`. For example: `puppet infrastructure help provision`.

Related information

[PostgreSQL WAL disk space](#) on page 210

The `max_slot_wal_keep_size` setting specifies the maximum allocated WAL disk space for each replication slot. This prevents the `pg_wal` directory from growing infinitely.

[Generate a token using puppet-access](#) on page 305

Use the `puppet-access` command 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 completed the steps outlined in the [Configure disaster recovery](#) section.

Important: The process outlined here isn't suitable if your installation is configured by the Puppet Enterprise Administration Module (PEADM). See [Provision and enable a replica for a PEADM installation](#).

1. Configure infrastructure agents to connect orchestration agents to the primary server.

- In the console, click **Node groups**, and in the **PE Infrastructure** group, select the **PE Agent > PE Infrastructure Agent** group.
- 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.
- On the **Classes** tab, find the **puppet_enterprise::profile::agent** class and specify these parameters:

Parameter	Value
manage_puppet_conf	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. This is the default value.
pcp_broker_list	Hostname for your primary server. Hostnames must include port 8142, for example <code>["PRIMARY.EXAMPLE.COM:8142"]</code> .
primary_uris	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> .
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> .

- Remove any values set for `pcp_broker_ws_uris`.
- Commit changes.
- Run Puppet on all agents classified into the **PE Infrastructure Agent** group.

- On the primary server, as the root user, run `puppet infrastructure provision replica <REPLICA NODE NAME> --enable`

Tip:

The default `replica --enable` command adds the replica to your PE Agent node group's server list, which causes all `Puppet.conf` files to include the new server. However, if you include the `--skip-agent-config` flag, the replica is added to the server list of the PE Infrastructure Agent node group (which is a child of the PE Agent node group); this, by extension, impacts only the `Puppet.conf` files on your infrastructure nodes (including compilers).

In installations with compilers, use the `--skip-agent-config` flag with `--enable` if you want to:

- Upgrade a replica without needing to run Puppet on all agents.
- Add disaster recovery to an installation without modifying the configuration of existing load balancers.
- Manually configure which load balancer agents communicate with in multi-region installations. See [Managing agent communication in multi-region installations](#) on page 261.

- Copy your secret key files from the primary server to the replica.

The secret key files are located at:

- `/etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json`
- `/etc/puppetlabs/orchestration-services/conf.d/secrets/orchestrator-encryption-keys.json`
- `/etc/puppetlabs/console-services/conf.d/secrets/keys.json`

Important: If you do not copy your secret key files onto your replica, the replica generates new secret key files when you promote it. This prevents you from accessing LDAP, and prevents services from accessing encrypted information in PE databases.

- Verify that the contents of the global layer Hiera file on the new replica, located at `/etc/puppetlabs/puppet/hiera.yaml`, match the contents of the global layer Hiera file on the primary server.
 - If necessary, update `hiera.yaml` on the replica to match `hiera.yaml` on the primary server.
 - If you use code to manage the contents of `hiera.yaml` on the primary server, ensure that the new replica is also classified to manage the contents of its own `hiera.yaml` file.
- Optional: Verify that all services running on the primary server are also running on the replica:
 - From the primary server, run `puppet infrastructure status --verbose` to verify that the replica is available.
 - 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.
- 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}'
```

7. 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 305

Use the `puppet-access` command to generate an authentication tokens from the command line of any workstation (Puppet-managed or not), without the need to SSH into the primary server.

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

[Back up your infrastructure](#) on page 847

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

[Running Puppet on nodes](#) on page 438

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 trigger a Puppet run.

Managing agent communication in multi-region installations

Typically, when you enable a replica by using `puppet infrastructure enable replica` or `puppet infrastructure provision replica --enable`, the configuration tool automatically sets the same communication parameters for all agents. In *multi-region installations*, with load balancers or compilers in multiple locations, you must manually configure agent communication settings so that agents fail over to the appropriate load balancer or compiler.

To skip automatically configuring which Puppet servers and PCP brokers agents communicate with, use the `--skip-agent-config` flag when you provision and enable a replica, for example:

```
puppet infrastructure provision replica example.puppet.com --enable --skip-agent-config
```

To manually configure which load balancer or compiler agents communicate with, use one of these options:

- CSR attributes
 1. For each node, include a CSR attribute that identifies the location of the node, for example `pp_region` or `pp_datacenter`.
 2. Create child groups off of the **PE Agent** node group for each location.
 3. In each child node group, include the `puppet_enterprise::profile::agent` class 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`.

Provision and enable a replica for a PEADM installation

Before you begin

Ensure you have completed the steps outlined in the [Configure disaster recovery](#) section.

The following outlines the varying processes to provision and enable a replica for standard, large and extra-large PEADM-configured installations.

Procedure

- A standard or large PEADM installation must use the `peadm::add_replica` plan to expand to a standard disaster recovery or large disaster recovery installation.
- An extra-large PEADM installation must first use the `peadm::add_database` plan to prepare a replica-postgresql node before using the `peadm::add_replica` to complete expansion to an extra-large disaster recovery installation. When running the `peadm::add_replica` plan, you must also set the `replica_postgresql_host` parameter to the database host you just added with the `peadm::add_database` plan.
- In a standard disaster recovery configuration, or extra-large disaster recovery configuration without compilers, you need to manually reset the PE agent node groups `puppet_enterprise::profile::agent` parameters with the new primary/replica addresses.

For more information see:

<https://github.com/puppetlabs/puppetlabs-peadm/blob/main/documentation/expanding.md>.

Promote a replica

If your primary server can't be restored, you can promote the replica to establish it 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, and replication between the primary server and replica could cause additional issues with the promotion process. If you still have access to the primary server, stop all PE services by running `systemctl stop puppet` and then `systemctl stop pe-*`.

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 promotion.

3. When promotion is complete, update any systems or settings that refer to the old primary server, such as PE client tool configurations, Code Manager hooks, and CNAME records.
4. Deploy updated configuration to nodes by running Puppet or waiting for the next scheduled run.

Note: In case of a failover, scheduled Puppet and task runs are rescheduled based on the last execution time.

5. 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.

After promotion, 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 438

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 trigger a Puppet run.

[Connect to a SAML identity provider](#) on page 290

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 enable_ha_failover` command detailed here leverages a built-in Bolt plan. To use this command, you must be able to connect using SSH from your current primary server to the failed primary server. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. Additionally, the tasks used by the plan must run as root, so specify the `--run-as root` flag with the command, as well as `--sudo-password` if necessary. For more information, see [Bolt OpenSSH configuration options](#).

Important: The process outlined here isn't suitable if your installation is configured by the Puppet Enterprise Administration Module (PEADM). See [Enable a new replica using a failed primary server for a PEADM installation](#).

By default, the `enable_ha_failover` plan uses its own RBAC user to perform the provision and enable commands. If you want to use a specific user instead, specify the RBAC parameters to the command.

To view all available parameters, use the `--help` flag. The logs for this and all `puppet infrastructure run` Bolt plans are located at `/var/log/puppetlabs/installer/bolt_info.log`.

To minimize the time required to enable a new replica from a failed primary server, the default plan attempts to provision the failed server as a replica by retaining the existing PuppetDB database. However, if the failed server has been offline for an extended period, the backlog of data may cause synchronization issues with the current primary, especially if the rate of new data generation is higher than PuppetDB's sync capacity. In such cases, consider using the alternative workflow to completely reinstall Puppet Enterprise and re-provision the replica. To activate this workflow, you can run the plan with `uninstall_workflow=true`.

Note: The alternative workflow takes longer as it involves copying the entire PuppetDB database to the node. This workflow also backs up the node's log history by saving the contents of `/var/log/puppetlabs` to `/var/log/puppetlabs_<timestamp>`.

To repurpose a failed primary server as a new replica, run the `enable_ha_failover` plan as follows:

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` — The architecture used in your environment, either `mono` (for a standard installation) or `mono-with-compile` (for a large installation). For `mono-with-compile`, you must specify either `skip_agent_config`, or **both** `agent_server_urls` and `pcp_brokers`.
 - `skip_agent_config` — Optional. Specifying this parameter with `topology=mono-with-compile` skips configuring `puppet.conf` on non-infrastructure agent nodes. This parameter is ignored when `topology=mono`.
 - `agent_server_urls` — Optional. This is the parameter used with `topology=mono-with-compile` to specify the `server_list` parameter in `puppet.conf` on all agent nodes. This parameter is ignored when `topology=mono`.
 - `pcp_brokers` — Optional. This is the parameter used with `topology=mono-with-compile` to list the PCP brokers for PXP agent's configuration file. This parameter is ignored when `topology=mono`.
- `dns_alt_names` — Optional. A comma-separated list of DNS alt names to add to the host's certificate.

- `rbac_account` — Optional. The RBAC account you want to use to run the provision and enable commands, instead of the built-in `enterprise_tasks` user.
- `rbac_password` — Optional. The password for the RBAC account you want to use to run the provision and enable commands.
- `replication_timeout_secs` — Optional. The number of seconds allowed to complete provisioning and enabling of the new replica before the command fails.
- `uninstall_workflow` — Optional. Use the uninstall/reinstall workflow instead of the default workflow.
- `force` — Skip some checks when running the plan.

For example:

```
puppet infrastructure run enable_ha_failover host=<FAILED_PRIMARY_HOSTNAME>
topology=mono
```

The failed primary server is repurposed as a new replica.

Enable a new replica using a failed primary server for a PEADM installation

To reuse an old primary server as a new replica on a PEADM-configured installation:

1. Uninstall PE on the old primary server: `puppet-enterprise-uninstaller -ypd`.
2. From your PEADM jump host, run the `peadm::add_replica` plan with `primary_host` set to your newly promoted primary server, and `replica_host` the old primary server that you just uninstalled.

When performing this in an extra-large disaster recovery environment, you must also supply the `replica_postgresql_host` parameter.

3. In a standard disaster recovery configuration, or an extra-large disaster recovery without compilers, you need to manually reset the PE agent node groups `puppet_enterprise::profile::agent` parameters with the new primary/replica addresses.

For more information see:

https://github.com/puppetlabs/puppetlabs-peadm/blob/main/documentation/automated_recovery.md

Forget a replica

Forgetting a replica removes the replica from classification and database state and purges the node.

Before you begin

Ensure you have a valid admin RBAC token and the replica you want to remove is permanently offline.

Run the `forget` command whenever a replica node is destroyed, even if you plan to replace it with a replica with the same name.

You can also follow this process if your replica is offline for an extended period. When the replica is offline, PostgreSQL Write-Ahead Log (WAL) files build up on the primary server, potentially consuming excessive disk space. To avoid this, you can run the `forget` command and then reprovision the replica.

1. On the primary server, as the root user, run `puppet infrastructure forget <REPLICA NODE NAME>`
2. If the replica node still exists, run `puppet-enterprise-uninstaller -y -p -d` to completely remove Puppet Enterprise from the node. This action helps to avoid security risks associated with leaving sensitive information in the PostgreSQL database and secret keys on a replica.

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 305

Use the `puppet-access` command 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 `puppet infrastructure status` shows errors on your replica after provisioning, you can reinitialize the replica. Reinitializing destroys and re-creates replica databases, except for PuppetDB. This process is usually quick because non-PuppetDB databases are relatively small.

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 enabled replica, 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 not operational or if you encounter replication errors on non-PuppetDB databases.

1. On the replica, as the root user, run `puppet infrastructure reinitialize replica`.
 - a) Optionally, you can reinitialize a single database with `puppet infrastructure reinitialize replica --db <DATABASE>`, replacing `<DATABASE>` with one of the following:
 - `pe-activity`
 - `pe-classifier`
 - `pe-orchestrator`
 - `pe-inventory`
 - `pe-rbac`
2. Follow prompts to complete the reinitialization. You can use the `-y` flag to bypass the prompts.

Accessing the console

The console is the web interface for Puppet Enterprise.

Use the console to:

- Manage node requests to join the Puppet deployment.
- Assign Puppet classes to nodes and groups.
- Run Puppet on specific groups of nodes.
- View reports and activity graphs.
- Browse and compare resources on your nodes.
- View package and inventory data.
- Manage console users and their access privileges.
- [Reaching the console](#) on page 265

The console is served as a website over SSL, on whichever port you chose when installing the console component.

- [Logging in](#) on page 266

Accessing the Puppet Enterprise (PE) console requires a username and password.

Reaching the console

The console is served as a website over SSL, on whichever port you chose when installing the console component.

Let's say your console server is `console.domain.com`. If you chose to use the default port (443), you can omit the port from the URL and reach the console by navigating to `https://console.domain.com`.

If you chose to use port 8443, you reach the console at `https://console.domain.com:8443`.

Remember: Always use the `https` protocol handler. You cannot reach the console over plain `http`.

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 Puppet Enterprise (PE) console requires a username and password.

If you are an administrator configuring or accessing the PE console for the first time, use the username and password you chose when you installed PE. Otherwise, get credentials from your site's administrator.

Because the console is your infrastructure's main control point, don't allow your browser to store the login credentials.

Generate a user password reset token

When users forget their passwords or lock themselves out of the console by providing incorrect credentials too many times, you must 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, 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 that is running console services.

1. On the node running console services (usually your primary server), log in as root.
2. To reset the console admin password, run:

```
puppet infrastructure console_password --password <MY_PASSWORD>
```

Troubleshooting PE admin account access

You might encounter these situations when trying to log in as the Puppet Enterprise (PE) admin user.

Multiple admin users

If your directory has multiple users with `admin` as their login name, the PE admin account can't log in.

PE admin locked out

If you are locked out of the PE admin account, ask another user with administrator access to [Generate a user password reset token](#) on page 266 for the admin user.

If there are no other users who can reset the admin user's password, you must SSH into the box and use curl commands to reset the directory service settings. For example, this curl command is for a box named centos7:

```
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"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Create a custom login disclaimer

You can add a custom banner to console login page. For example, you can add a disclaimer about authorized or unauthorized use of private information found in the console.

These steps explain how to use a `disclaimer.txt` file for your custom disclaimer. You can also use the RBAC API [Disclaimer endpoints](#) on page 357 to configure the disclaimer without needing to reference a specific file location on disk.

1. Create a `disclaimer.txt` file containing the disclaimer content.
2. Place the file in `/etc/puppetlabs/console-services`
If you want to store the file somewhere else, you can change the disclaimer file path in the console by configuring `puppet_enterprise::profile::console::disclaimer_content_path`
3. Log in to the console to test the new banner.

Related information

[Configure the PE console and console-services](#) on page 230

You can configure the behavior of the console and the `console-services` service.

Require LDAP group membership to log in

You can use the `exclude-groupless-ldap-users` setting to prevent LDAP users with no group bindings from logging in and creating Puppet Enterprise (PE) accounts. This setting is disabled by default.

1. On your primary server, navigate to `/etc/puppetlabs/console-services/conf.d/` and create a new `.conf` file at this location.
2. Paste the following into the `.conf` file:

```
rbac: {
  feature-flags: {
    exclude-groupless-ldap-users: true
  }
}
```

3. To merge this setting into your RBAC configuration, run Puppet on your primary server: `puppet agent -t`

Related information

[Managing access](#) on page 268

Role-based access control (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.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

Managing access

Role-based access control (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 users appropriate levels of access and capability. For example, you can use permissions to allow users to:

- Grant password reset tokens to other users who have forgotten their passwords.
- Edit a local user's metadata.
- Deploy Puppet code to specific environments.
- Edit class parameters in a node group.

You can do access control tasks in the console or with the RBAC API.

- [User permissions and user roles](#) on page 269

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 with that role can or can't do within Puppet Enterprise (PE).

- [Creating and managing local users and user roles](#) on page 278

Role-based access control (RBAC) in Puppet Enterprise (PE) lets you manage users—what they can and can't create, edit, or view—in an organized, high-level way that is 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 directly to users in PE, only to user roles. You then assign roles to users.

- [LDAP authentication](#) on page 281

Connect PE to an external Lightweight Directory Access Protocol (LDAP) directory service and manage permissions with role-based access control (RBAC).

- [SAML authentication](#) on page 288

Connect to a Security Assertion Markup Language (SAML) identity provider, like Microsoft ADFS or Okta, to log in to PE with single sign-on (SSO) or multifactor authentication (MFA).

- [Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

- [RBAC API](#) on page 310

Use the RBAC API to manage users, user groups, roles, permissions, tokens, password, and LDAP or SAML connections.

- [Activity service API](#) on page 368

The activity service records changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles. Use the activity service API to query event data.

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 with that role can or can't do within Puppet Enterprise (PE).

When you add users to PE, they don't have permission to do anything until you associate them with a user role, either explicitly through role assignment or implicitly through group membership and role inheritance. When a user is assigned to a role (or inherits a role from a group), they receive all the permissions from that role.

There are five default user roles:

Administrators

Can manage users and permissions, create and modify node groups and other objects.

Administrators have all permissions assigned to them by default.

Operators

Can create and modify node groups and other objects.

Viewers

Can view, but can't modify, objects in the console.

Code Deployers

Can synchronize code from version control systems to Puppet Server.

Project Deployers

Can deploy projects and run project tasks and plans.

You can also create custom roles. For example, you might want to create a user role that grants users permission to view but not edit a specific subset of node groups. Or you might want to divide up administrative privileges so that one user role is able to reset passwords while another can edit roles and create users.

Permissions are additive. If a user is associated with multiple roles, that user is able to perform all of the actions described by all of the permissions on all of the applied roles.

Structure of user permissions

User permissions are structured as a triple of *type*, *permission*, and *object*.

- **Types:** Any thing that can be acted on in Puppet Enterprise (PE), such as node groups, users, or user roles.
- **Permissions:** What you can do with each type, such as create, edit, or view.
- **Objects:** Specific instances of types.

For example, here are two sets of permission triples for the Administrators user role:

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, then the permission applies to all objects of the specified type. In those cases, the object is All. This is denoted by "*" in the RBAC 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.

Reference: User permissions and names

This reference describes the permissions granted to the five default Puppet Enterprise (PE) user roles, as well as the *display name* and *system name* for each type and permission.

Permissions vary by type. Permission scope can, sometimes, be refined by an object specification. For an explanation of these terms, refer to [Structure of user permissions](#) on page 269.

Each type and permission has a *display name*, which is the name you see in the PE console, and a *system name*, which is the name used in the RBAC API. The **Permissions for default roles** table uses display names. Refer to the **Display names and system names** table to find the corresponding system name for each display name.

Permissions for default roles

This table lists permissions granted to the [default PE user roles](#), a description of what is allowed by each permission, any object specification or node group inheritance conditions, and which roles (if any) the permissions are assigned to by default.

Type	Permission	Definition	Roles
Certificate request	Accept and reject	Accept and reject certificate signing requests. Object must always be "*".	<ul style="list-style-type: none"> Administrators Operators
Configuration	View and edit	View and edit configuration, such as the disclaimer message on the PE console login page.	Administrators
Console	View	View the PE console. Object must always be "*".	<ul style="list-style-type: none"> Administrators Operators Viewers
Directory service	View, edit, and test	View, edit, and test directory service settings. Object must always be "*".	Administrators
Job orchestrator	Start, stop and view jobs	Start and stop jobs and tasks, view jobs and job progress, view an inventory of nodes that are connected to the PCP broker.	<ul style="list-style-type: none"> Operators Viewers Project Deployers

Type	Permission	Definition	Roles
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 descendants of the node group.	<ul style="list-style-type: none"> Administrators Operators
Node groups	Edit child group rules	Edit the rules of descendants 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 descendants of the node group.	<ul style="list-style-type: none"> Administrator Operators
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 descendants of the node group.	<ul style="list-style-type: none"> Administrators Operators
Node groups	Edit configuration data	Edit parameterized configuration data on a node group. This permission is inherited by all descendants of the node group.	<ul style="list-style-type: none"> Administrators Operators

Type	Permission	Definition	Roles
Node groups	Edit parameters and variables	Edit the class parameters and variables of a node group's classes. This permission is inherited by all descendants of the node group.	• Administrators • Operators
Node groups	Set environment	Set the environment of a node group. This permission is inherited by all descendants of the node group.	• Administrators • Operators
Node groups	View	See all attributes of a node group, including the values of class parameters and variables. This permission is inherited by all descendants of the node group.	• Administrators • Operators • Viewers
Nodes	Edit node data from PuppetDB	Edit node data imported from PuppetDB. Object must always be " <code>*</code> ".	Administrators
Nodes	View node data from PuppetDB	View node data imported from PuppetDB. Object must always be " <code>*</code> ".	Administrators
Nodes	View sensitive connection information in inventory service	View sensitive parameters stored in the inventory service for a connection, such as user credentials. Object must always be " <code>*</code> ".	Administrators
Nodes	Add and delete connection information from inventory service	Add new connections to the inventory service and delete existing connections.	Administrators
Plans	Run plans	Run specific plans on all nodes.	Administrators

Type	Permission	Definition	Roles
Projects	Deploy projects	Not used.	<ul style="list-style-type: none"> • Administrators • Project Deployers
Projects	Run tasks and plans from projects	Not used.	<ul style="list-style-type: none"> • Administrators • Project Deployers
Puppet agent	Run Puppet on agent nodes	Trigger a Puppet run from the console or orchestrator. Object must always be " <code>*</code> ".	<ul style="list-style-type: none"> • Administrators • Operators
Puppet environment	Deploy code	Deploy code to a specific PE environment.	<ul style="list-style-type: none"> • Administrators • Operators • Code Deployers
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.	Administrators
Scheduled jobs	Delete another user's scheduled jobs	Delete scheduled jobs created by the user instance specified in the permission. This can be granted per user.	Administrators
Tasks	Run tasks	Run specific tasks on all nodes, nodes in a selected node group, or nodes matching a PQL query.	Administrators
<p>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.</p>			

Type	Permission	Definition	Roles
User groups	Delete	Delete a user group. This can be granted per group.	Administrators
User groups	Import	Import groups from the directory service for use in RBAC. Object must always be " * ".	Administrators
User roles	Create	Create new roles. Object must always be " * ".	Administrators
User roles	Edit	Edit and delete a role. Object must always be " * ".	Administrators
User roles	Edit members	Change which users and groups a role is assigned to. This can be granted per role.	Administrators
Users	Create	Create new local users. Object must always be " * ".	Administrators
Tip: This permission is for local users. Remote users are "created" when that user authenticates for the first time with RBAC.			
Users	Edit	Edit local user data (such as names or email addresses) and delete local or remote users from PE. This can be granted per user.	Administrators

Type	Permission	Definition	Roles
Users	Reset password	<p>Grant password reset tokens to users who have forgotten their passwords.</p> <p>Granting a password reset token also reinstates a user who has been revoked.</p> <p>This can be granted per user.</p>	Administrators
Users	Revoke	<p>Revoke or disable a user, so the user can no longer authenticate and use the console, node classifier, or RBAC API.</p> <p>This permission also includes the ability to revoke a user's authentication tokens.</p> <p>This can be granted per user.</p>	Administrators

Display names and system names

Each type and permission has a *display name* and a *system name*. The display name is the name you see in the PE console. The system name is the name used with the [RBAC API](#) on page 310. This table provides the display name and system name for each type and corresponding permissions. Types are listed multiple times if there are multiple permissions associated with that type.

Type display name	Type system name	Permission display name	Permission system name
Certificate requests	cert_requests	Accept and reject	accept_reject
Configuration	configuration	View	view
Configuration	configuration	Edit	edit
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

Type display name	Type system name	Permission display name	Permission system name
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 338

You add permissions to roles to control what users can access and do in PE. Use the `permissions` endpoints to get information about objects you can create permissions for, what types of permissions you can create, and whether specific users can perform certain actions.

Working with node group permissions

Node groups in the node classifier are structured hierarchically; therefore, node group permissions are inherited. Users with specific permissions on a node group implicitly receive those 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 node group allows the user to set the environment for that node group and all of the node group's children. However, assigning `Edit child group rules` to a node 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 parent node group, while other users can be given permissions to modify the group's children without being able to affect the parent group.

Due to the hierarchical nature of node groups, if a user is given a permission on the default node group (`All nodes`), this is functionally equivalent to giving them that permission on all objects of the node group type ("*").

Related information

[Structure of user permissions](#) on page 269

User permissions are structured as a triple of *type*, *permission*, and *object*.

Best practices for assigning permissions

Working with user permissions requires delicacy. You don't want to unintentionally escalate users' roles by granting them excessive permissions, but you also don't want to hamper them in their day-to-day duties. The following sections describe some strategies and requirements for setting permissions in Puppet Enterprise (PE).

Grant edit permissions to users with create permissions

Creating objects doesn't automatically grant the creator permission to view those objects. Therefore, users who have permission to create an object (such as roles) must also be given permission to edit the same object. Otherwise, they can't see the object they create. When you grant permission to create an object, we recommend that you also grant permission to edit all objects of the type that they have permission to create.

For example, to allow a user to create roles and also view/edit the roles they create, we recommend assigning these permission sets:

Type	Permission	Object
User roles	Create	A specific object or all objects of this type ("*")
User roles	Edit	All (or "*")

Tip: If you also want the role creator to be able to assign users to the role or view role membership, make sure you also grant the `Edit members` permission for all objects ("*").

As another example, to allow a user to create user records and also see the user records they create, we recommend the following:

Type	Permission	Object
Users	Create	A specific object or all objects of this type ("*")
Users	Edit	All (or "*")

Related information:

- [Structure of user permissions](#) on page 269
- [Reference: User permissions and names](#) on page 270

Least-privilege model: Avoid overly-permissive permissions

Operators, one of the default PE roles, have many of the same permissions as Administrators. However, we've intentionally limited this role's ability to edit user roles. This way, users with the Operators role can do many of the same things as Administrators, but they can't edit (or enhance) their own permissions.

Similarly, when you're editing permission or creating your own roles, avoid granting users more permissions than necessary. For example, if users have the `roles:edit:*` permission, this allows them to add the `node_groups:view:*` permission to the roles they belong to, and, subsequently, see all node groups. Take care that permissions you've granted don't have the potential to allow a user to view or change something you don't want them to view or change.

Grant edit directory service permissions sparingly

The directory service password is not redacted when a user requests directory service settings through the RBAC API. Make sure the `directory_service:edit:*` permission is **only** granted to users who are allowed see the directory service password and other settings.

Grant reset password permissions along with other password permissions

The `users:reset_password:<INSTANCE>` permission allows a user to issue a password reset token to a user who forgot their password. However, this permission also reinstates revoked users when the password reset token is used. Therefore, make sure the users you allow to reset passwords are also allowed to revoke and reinstate users. Otherwise, you might have unauthorized users reinstating revoked users.

Creating and managing local users and user roles

Role-based access control (RBAC) in Puppet Enterprise (PE) lets you to manage users—what they can and can't create, edit, or view—in an organized, high-level way that is 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 directly to users in PE, only to user roles. You then assign roles to users.

In addition to user records that you create, PE includes two default user records:

- **Administrator:** A user that has the Administrator role applied by default. This means this user has every permission. You can revoke the Administrator user in situations where users are managed through a directory service, like LDAP.
- **API User:** Used for service-to-service authentication within PE. You can't use it with the standard login, and you can't revoke it. It is only available through certificate-based authentication. The RBAC *allow list* identifies the certificates (by certname) that you can use for API User authentication.

Remember: All user records you create 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. To update the algorithm to argon2id (only for non-FIPS enabled systems) or to configure password algorithm parameters, refer to [Configure the password algorithm](#) on page 227. 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 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 user name for the user.
4. Click **Add local user**.

Give a user access to the PE console

When you create local users, you need to send them a password reset token that allows them to log in to PE 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 user role

Puppet Enterprise (PE) includes five default roles. You can also create your own roles.

For information about the five default roles, refer to [User permissions and user roles on page 269](#).

Users with the appropriate permissions, such as Administrators, can create custom roles. To avoid unintentional privilege escalation, make sure the only users who can edit user roles are those who have all permissions (meaning Administrators). For more information, refer to [Best practices for assigning permissions](#) on page 277.

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 access to Puppet Enterprise (PE) actions.

Before you begin

Review [User permissions and user roles](#) on page 269 for 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 277

Working with user permissions requires delicacy. You don't want to unintentionally escalate users' roles by granting them excessive permissions, but you also don't want to hamper them in their day-to-day duties. The following sections describe some strategies and requirements for setting permissions in Puppet Enterprise (PE).

Add a user to a user role

When you add a user to a role, the user gains the permissions you assign to that role. A user can't do anything in PE until they have been assigned to at least one role. If users are assigned to multiple roles, they get all permissions from all roles they are assigned to.

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

When you remove a user from a role, the user loses the permissions associated with that role. If you remove all roles from a user, the user can't do anything in PE until they are assigned to at least one 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 or reinstate user access

If you want to stop a user from accessing PE without deleting their account, you can revoke the user. Users are automatically revoked if they have too many incorrect password attempts. This is also referred to as locking a user's account. You can use these steps to revoke users or reinstate revoked users.

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 reinstate a revoked user, click **Reinstate user access**.

Change account expiration settings

You can specify the number of days before an inactive user's account is automatically revoked. You can also specify how often Puppet Enterprise (PE) checks for idle user accounts.

`rbac_account_expiry_days`

The `rbac_account_expiry_days` parameter is a positive integer specifying the duration, in days, before an inactive user account expires. If a user (who isn't a superuser) doesn't log in to the PE console at least once during the specified period, their user's access is automatically revoked.

The default value is undefined, meaning no expiration limit. To activate this setting in the console, specify a value of 1 or greater for the `rbac_account_expiry_days` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** node group. The value corresponds to the number of days an account can be idle before being revoked. For example, 30 would be 30 days.

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.

`rbac_account_expiry_check_minutes`

The `rbac_account_expiry_check_minutes` parameter is a positive integer that specifies how often, in minutes, PE checks for idle user accounts. The default value is 60 minutes.

To change this setting in the console, set a value (representing a number of minutes) of the `rbac_account_expiry_check_minutes` parameter in the `puppet_enterprise::profile::console` class of the **PE Infrastructure** group.

Related information:

- [Configure RBAC and token-based authentication settings](#) on page 224
- [How to configure PE](#) on page 211

Delete a user

You can delete a user through the Puppet Enterprise (PE) console. This deletes only the user's PE account. It does not delete 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 [Activity service API](#) on page 368.

Tip: If you delete a user and then create a user with the same full name and login, PE issues a new user ID for the new user record. When this happens, requests to the [Activity service API](#) on page 368 about this ID return information from both the deleted user record and the new user record. However, in the PE console, the new user record's **Activity** tab does not display information about the deleted user's account.

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.

You can't delete users with superuser privileges. The **Remove** button is not available when viewing these users.

Delete a user role

You can delete a user role through the Puppet Enterprise (PE) console.

When you delete a user role, any users assigned to that role are no longer assigned to it. Therefore, those users lose the permissions that the role gave them. This can impact their access to PE if they are not assigned other 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**.

LDAP authentication

Connect PE to an external Lightweight Directory Access Protocol (LDAP) directory service and manage permissions with role-based access control (RBAC).

- [Connecting LDAP external directory services to PE](#) on page 281

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 287

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.

Related information

[Require LDAP group membership to log in](#) on page 267

You can use the `exclude-groupless-ldap-users` setting to prevent LDAP users with no group bindings from logging in and creating Puppet Enterprise (PE) accounts. This setting is disabled by default.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

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. To update the algorithm to argon2id (only for non-FIPS enabled systems) or to configure password algorithm parameters, refer to [Configure the password algorithm](#) on page 227. 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.

Display name	System name (for RBAC API)	Example Active Directory settings	Example OpenLDAP settings
Directory name	display_name	My Active Directory	My Open LDAP Directory
Login help (optional)	help-link	https://myweb.com/ldaploginhelp	https://myweb.com/ldaploginhelp
Hostname	hostname	myhost.delivery.example.com	myhost.delivery.example.com
Port	port	389 (or 636 for LDAPS)	389 (or 636 for LDAPS)
Lookup user (optional)	login	cn=queryuser,cn=Users,dc=puppetlabs,dc=puppet	cn=queryuser,cn=Users,dc=puppetlabs,dc=puppet
Lookup password (required if the lookup user is specified)	password	The lookup user's password.	The lookup user's password.
Connection timeout (seconds)	connect_timeout	10	10
Connect using:	ssl, start_tls	SSL	StartTLS
Validate the hostname?	ssl_hostname_validation	Default is yes	Default is yes
Allow wildcards in SSL certificate?	ssl_wildcard_validation	Default is no	Default is no
Base distinguished name	base_dn	dc=puppetlabs,dc=com	dc=puppetlabs,dc=com
User login attribute	user_lookup_attr	SAMAccountName	cn
User email address	user_email_attr	mail	mail

Display name	System name (for RBAC API)	Example Active Directory settings	Example OpenLDAP settings
User full name	user_display_name_attr	displayName	displayName
User relative distinguished name (optional)	user_rdn	cn=users	ou=users
Group object class	group_object_class	group	groupOfUniqueNames
Group membership field	group_member_attr	member	uniqueMember
Group name attribute	group_name_attr	name	displayName
Group lookup attribute	group_lookup_attr	cn	cn
Group relative distinguished name (optional)	group_rdn	cn=groups	ou=groups
Turn off LDAP_MATCHING_RULE_IN_CHAIN?	disable_ldap_matching_in_chain	Default is no	Default is no
Search nested groups?	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 286.

Validate the hostname?

Select **Yes** to verify that the Directory Services hostname used to connect to the LDAP server matches the hostname on the SSL certificate. This option is not available when you choose to connect to the external directory using plain text.

Allow wildcards in SSL certificate?

Select **Yes** to allow a connection to a Directory Services server with a SSL certificates that use a wildcard (*) specification. This option is not available when you choose to connect to the external directory using plain text.

Base distinguished name

When PE constructs queries to your external directory (for example to look up user groups or users), the queries consist of the relative distinguished name (RDN) (optional) + the base distinguished name (DN), and are then filtered by lookup/login attributes. For example, if PE wants to authenticate a user named Bob who has the RDN `ou=bob,ou=users`, it sends a query in which the RDN is concatenated with the DN specified in this field (for example, `dc=puppetlabs,dc=com`). This gives a search base of `ou=bob,ou=users,dc=puppetlabs,dc=com`.

The base DN that you provide in this field specifies where in the directory service tree to search for groups and users. It is the part of the DN that all users and groups that you want to use have in common. It is commonly the root DN (example `dc=example,dc=com`) but in the following example of a directory service entry, you could set the base DN to `ou=Puppet,dc=example,dc=com` because both the group and the user are also under the organizational unit `ou=Puppet`.

Example directory service entry:

```
# 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 347

Replace current directory service connection settings. You can update the settings or disconnect the service (by removing all 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 281. 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.

- 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.

- 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.

- Locate the group that you wish to delete.
- 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 LDAP external directory service but not from PE, the user can no longer log in. However, any generated tokens or existing console sessions remain valid until they expire or are revoked by automatic LDAP synchronization, which is controlled by the `ldap_sync_period_seconds` parameter. For information about modifying this parameter, see [Console and console-services parameters](#) on page 134.

To manually invalidate the user's tokens or sessions, you must 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.

Related information

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

[Change the default token lifetime](#) on page 308

Tokens have a default authentication lifetime of one hour, but this default value can be adjusted in the console. You can also change the maximum permitted lifetime, which defaults to 10 years.

SAML authentication

Connect to a Security Assertion Markup Language (SAML) identity provider, like Microsoft ADFS or Okta, to log in to PE with single sign-on (SSO) or multifactor authentication (MFA).

- [Connect a SAML identity provider to PE](#) on page 289

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.

- [Connect Microsoft ADFS to PE](#) on page 295

Connect to Microsoft Active Directory Federation Services (ADFS) on a Windows server, enabling users to log in to PE using their ADFS credentials.

- [Connect Okta to PE](#) on page 300

Connect to Puppet Enterprise (PE) to Okta so that users can log in to PE with their Okta credentials.

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 352 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** and note the following values:
 - SAML metadata URL
 - SAML assertion consumer service (acs) URL
 - SAML Single Logout URL
 - Signing and Encryption Certificate
3. Copy the URLs and certificate so you can add them to your identity provider configuration.

Tip: Copy the entire certificate, including the begin and end tags, into its own file.

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. Input the configuration information.

The [SAML configuration reference](#) on page 291 explains what to input in each field and which fields are optional.

Important: Make sure the **Organization URL** is a valid, well-formed URL. Supplying an invalid value does not produce a field validation error, but it does cause the following error at login: `Invalid settings: organization_not_enough_data`.

4. Commit changes.

Related information

[Promote a replica](#) on page 262

If your primary server can’t be restored, you can promote the replica to establish it as the new, permanent primary server.

[Troubleshooting SAML connections](#) on page 864

There are some common issues and errors that can occur when connecting a SAML identity provider to PE, such as failed redirects, rejected communications, and failed group binding.

Generate a token in the console

Use the console to generate an authentication token that you can use to access PE APIs. If SAML is configured, you must have a token to use CLI tools, such as 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 Puppet Enterprise (PE) and your SAML identity provider to enable SSO or MFA. All fields are required unless otherwise noted.

Setting name	System name (for RBAC API)	Definition
Allow duplicated attribute name?	allow_duplicated_attribute	Boolean value indicates whether PE allows duplicate attribute names in the attribute statement. Default is true.
Display name	display_name	A required string that identifies the IdP used for SSO or MFA login, such as Corporate Okta.
Identity provider entity ID	idp_entity_id	A URL string identifying your IdP, such as https://sso.example.info/entity. Your IdP's configuration has this information.
Identity provider SLO response URL	idp_slo_response_url	Optional, unless required by IdP or SAML configuration. An optional URL specifying an alternative location for SLO handling. Defaults to the SLO URL. For example, https://ipd.example.com/SAML2/SLO-response.
Identity provider SLO URL	idp_slo_url	The URL to which PE sends the single logout request (SLO), such as https://ipd.example.com/SAML2/SLO. Your IdP configuration has this information.

Setting name	System name (for RBAC API)	Definition
Identity provider SSO URL	idp_sso_url	The URL to which PE sends authentication messages, such as <code>https://idp.example.org/SAML2/SSO</code> . Your IdP configuration has this information.
IdP certificate	idp_certificate	The public x509 certificate of the identity provider, in PEM format. PE uses the certificate to decrypt messages from the IdP and validate signatures. For example: -----BEGIN CERTIFICATE----- MIIGADCCA +igAwIBAgIBAjANBgkqhkiG9w0BAQsFADB ... STkGww== -----END CERTIFICATE-----
Name ID encrypted?	name_id_encrypted	Optional, unless required by IdP or SAML configuration. Boolean value indicates whether you want PE to encrypt the name-id in the logout request. Default is <code>true</code> .
Organizational language	organizational_lang	The standard abbreviation for the preferred spoken language at your organization, such as <code>en</code> for English.
Organization display name	organizational_display_name	An alternative display name for your organization
Organization name	organizational_name	The official name of your organization.
Organization URL	organizational_url	The URL for your organization.
Requested authentication context	requested_auth_context	Comma-separated list of authentication contexts indicating the type of user authentication PE suggests to the IdP. Authentication types are defined in the <code>urn:oasis:names:tc:SAML:2.0:ac:classe</code> namespace of the SAML specification. For example: <code>urn:oasis:names:tc:SAML:2.0:ac:classe</code> or <code>urn:oasis:names:tc:SAML:2.0:ac:classe</code>

Setting name	System name (for RBAC API)	Definition
Requested authentication context comparison	requested_auth_context_comparison	<p>Indicates to the IdP the strength of the authentication context PE provides. Choose one of the following:</p> <ul style="list-style-type: none"> 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. exact: The requested authentication context comparison must be an exact match with one of the contexts PE provides. better: The requested authentication context comparison must be higher than the context PE provides.
Require encrypted assertions?	want_assertions_encrypted	Boolean value indicates whether you want the IdP to encrypt the assertion messages it sends to PE. Default is true.
Require name ID encrypted?	want_name_id_encrypted	Boolean value indicates whether you want the IdP to encrypt the name-id field in messages it sends to PE. Default is true.
Require signed assertions?	want_assertions_signed	Boolean value indicates whether you want the IdP to cryptographically sign assertion elements it sends to PE. Default is true.
Require signed messages?	want_messages_signed	Boolean value indicates whether you want the IdP to cryptographically sign all responses, logout requests, and logout response messages it sends to PE. Default is true.

Setting name	System name (for RBAC API)	Definition
Signature algorithm	signature_algorithm	Indicates which signing algorithm PE uses to sign messages. Choose one of the following: <ul style="list-style-type: none"> rsa-sha256 (default) rsa-sha1 dsa-sha1 rsa-sha384 rsa-sha512
Sign authentication requests?	authn_request_signed	Optional, unless required by IdP or SAML configuration. Boolean value indicates whether you want PE to cryptographically sign authentication request it sends to the IdP. Default is true.
Sign logout requests?	logout_request_signed	Optional, unless required by IdP or SAML configuration. Boolean value indicates whether you want PE to cryptographically sign logout requests it sends to the IdP. Default is true.
Sign logout response?	logout_response_signed	Optional, unless required by IdP or SAML configuration. Boolean value indicates whether you want PE to cryptographically sign logout responses it sends to the IdP. Default is true.
Sign metadata?	sign_metadata	Boolean value indicates whether you want PE to cryptographically sign the metadata provided for the IdP configuration. Default is true.
Support contact email address	support_email	The email address of the main support contact at your organization.
Support contact name	support_name	The name of the main support contact at your organization.
Technical contact email address	technical_support_email	The email address of the main technical contact at your organization.
Technical contact name	technical_support_name	The name of the main technical contact at your organization.

Setting name	System name (for RBAC API)	Definition
User display name attribute binding	user_display_name_attr	Identifies the attribute that maps to the user's displayable name, such as First name Last name.
User email attribute binding	user_email_attr	Identifies the attribute that maps to the user's email address.
User group lookup attribute binding	group_lookup_attr	Identifies the attribute that maps to the set of user groups a user belongs to.
User lookup attribute binding	user_lookup_attr	Identifies the attribute that maps to the login value users provide on the login page.
Validate xml?	want_xml_validation	Boolean value indicates whether you want PE to validate all xml statements it receives from your IdP, because invalid xml might cause security issues. Default is true.

Connect Microsoft ADFS to PE

Connect to Microsoft Active Directory Federation Services (ADFS) on a Windows server, enabling users to log in to PE using their ADFS credentials.

Note: This setup was tested using Windows Server 2019.

To connect ADFS to PE, add PE certificates to ADFS and configure SSO for ADFS in the PE console. Then, add PE as a relying trust party in ADFS, configure rules and groups, and add RBAC permissions for ADFS users.

1. [Add PE certificates to the ADFS server](#) on page 295
2. [Connect to ADFS in the PE console](#) on page 296
3. [Add the Relying Party Trust for PE to ADFS](#) on page 298
4. [Disable certificate revocation checking](#) on page 299
5. [Configure the Claim Issuance Policy in ADFS](#) on page 299
6. [Configure an RBAC group and role in PE](#) on page 299
7. [Test your SSO connection](#) on page 300

Related information

[Connect a SAML identity provider to PE](#) on page 289

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.

Add PE certificates to the ADFS server

To ensure ADFS trusts the certificates PE uses to sign requests, add the Puppet CA certificates to the Trusted Root CA store on the ADFS server. There can be one or two certificates to import, depending on which version of PE you upgraded from.

1. On your primary server, retrieve the certificates:

```
cat /etc/puppetlabs/puppet/ssl/certs/ca.pem
```

2. Depending on how many certificates appear, do one of the following:

- One certificate – copy the certificate text and paste it into a .cer file on your ADFS server. Then, import the certificate into the Trusted Root Certification Authorities store.
- Two certificates – export the certificates with this command:

```
openssl pkcs12 -export -nokeys -in /etc/puppetlabs/puppet/ssl/certs/
ca.pem -out ~/ca.pfx -passout pass
```

Copy the resulting ca.pfx file to your ADFS server, then import it into the Trusted Root Certification Authorities store. The file has no password. The two certificates appear after importing the file.

Connect to ADFS in the PE console

Use the PE console to connect ADFS.

1. In the console, on the **Access control** page, click the **SSO** tab.
2. Click **Configure**.
3. Input the configuration information as described in the [ADFS configuration reference](#) on page 296. Make sure to complete the **Organization** and **Contacts** sections.
4. Commit changes.

ADFS configuration reference

Configure ADFS in the PE console with these settings and values.

ADFS configuration values

In the PE console, configure these values in the **Identity provider information** and **Service provider configuration options** sections of the SSO configuration page.

Setting	Maps to	ADFS configuration value
Display name	display_name	Example: "ADFS"
Identity provider entity ID	idp_entity_id	An HTTP or HTTPS URL indicating the ADFS Identifier. To find your URL, in the ADFS Microsoft Management Console, click Edit Federation Service Properties . Example: "http://<federation service name>/adfs/services/trust"
Identity provider SSO URL	idp_sso_url	The ADFS Single Sign On URL. To find your SSO URL, in the ADFS Microsoft Management Console, navigate to ADFS > Service > Endpoints . Under Token Issuance , in the Type column, click the endpoint that specifies SAML 2.0/WS-Federation . Example: "https://<federation service name>/adfs/ls/"
Identity provider SLO URL	idp_slo_url	The ADFS Single Sign On URL with ?wa=wsignin1.0 added to the end. Example: "https://<federation service name>/adfs/ls/?wa=wsignin1.0"

Setting	Maps to	ADFS configuration value
Identity provider SLO response URL	idp_slo_response_url	The same as the ADFS SLO URL. Example: "https://<federation service name>/adfs/ls/?wa=wsignin1.0"
IdP certificate	idp_certificate	The ADFS Token Signing certificate. To get the certificate, run this PowerShell script on your ADFS server:
		<pre>\$cert = Get-AdfsCertificate - CertificateType Token-Signing ? IsPrimary -eq \$true \$oPem = New-Object System.Text.StringBuilder \$oPem.AppendLine("-----BEGIN CERTIFICATE-----") \$oPem.AppendLine([System.Convert]::ToBase64String(\$cer \$oPem.AppendLine("-----END CERTIFICATE-----") \$oPem.ToString() out-file ./ adfs_token_signing.pem</pre>
		Example: -----BEGIN CERTIFICATE----- MIIGADCCAiigAwIBAgIBAjANBgkqhki ... STkGww== -----END CERTIFICATE-----
Name ID encrypted?	name_id_encrypted	true
Sign authentication requests?	authn_request_signed	true
Sign logout response?	logout_response_signed	false
Sign logout requests?	logout_request_signed	true
Require signed messages?	want_messages_signed	false
Require signed assertions?	want_assertions_signed	false
Sign metadata?	sign_metadata	true
Require encrypted assertions?	want_assertions_encrypted	true

Setting	Maps to	ADFS configuration value
Require name ID encrypted?	want_name_id_encrypted	
Requested authentication context	requested_auth_context	urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtected
Requested authentication context comparison	requested_auth_context_comparison	
Allow duplicated attribute name?	allow_duplicated_attributes	attribute_name
Validate xml?	want_xml_validation	true
Signature algorithm	signature_algorithm	rsa-sha256

Attribute binding values for ADFS

In the PE console, add these values in the **Attribute binding** section of the SSO configuration page.

Attribute binding value	ADFS value
User	http://schemas.xmlsoap.org/claims/CommonName
Email	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
Display name	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
Groups	http://schemas.xmlsoap.org/claims/Group

Related information

[Attribute binding](#) on page 289

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.

Add the Relying Party Trust for PE to ADFS

Add PE to ADFS as a Relying Party Trust using a metadata address, allowing ADFS to recognize and communicate with PE as the service provider. Use the PE console to retrieve the metadata URL, then add it to ADFS using the ADFS Management console.

1. In the PE console, on the **Access Control** page, click the **SSO** tab, click **Show configuration information**, and copy the **SAML Metadata URL**.
2. In the ADFS Management console, click **Relying Party Trusts > Add Relying Trust Party > Claims aware**.
3. When the wizard opens, click **Start**.
4. Select **Import data about relying party published online or on a local network** and enter the **SAML Metadata URL**, then click **Next**.
5. Enter a **Display name** for your PE server, taking note of the name to refer to later, then click **Next**.
6. Accept the defaults for the **Access Control Policy** and click **Next**.
7. On the **Ready to Add Trust** page, click **Next**.

- On the **Finish** page, uncheck **Configure claims issuance policy for this application** and click **Close**.

Disable certificate revocation checking

ADFS can't look up the certificate revocation status because certificates from PE don't include CRL information. Use PowerShell to disable certificate revocation checking so ADFS doesn't perform certificate revocation checks on the relying party trust, resulting in trust failures.

- In PowerShell, display the names for all relying party trusts:

```
Get-AdfsRelyingPartyTrust | ft Name
```

- Find the trust with the display name you selected for your PE server.
- Determine the status of the revocation check for the PE trust:

```
Get-AdfsRelyingPartyTrust -Name <DISPLAY NAME> | ft
EncryptionCertificateRevocationCheck, SigningCertificateRevocationCheck
```

- If the encryption and signing certificate revocation checks show anything other than None, disable checking:

```
Get-AdfsRelyingPartyTrust -Name <DISPLAY NAME> | Set-
AdfsRelyingPartyTrust -SigningCertificateRevocationCheck None -
EncryptionCertificateRevocationCheck None
```

Configure the Claim Issuance Policy in ADFS

Add rules to the Claims Issuance Policy so it can send the correct LDAP attribute and user group information to PE.

Tip: In ADFS, a claim is the same thing as an assertion, and the Claims Issuance Policy defines what pieces of information about a user go where in a claim.

- In the ADFS Management console, click **Relying Party Trusts**.
- Select the PE trust you created and click **Edit Claim Issuance Policy**.
- Add a rule to send LDAP attributes as claims:

- Claim rule template:** Send LDAP Attributes as Claims
- Claim rule name:** LDAP Attributes
- Attribute store:** Active Directory LDAP attribute mappings

In the LDAP attribute mapping table, select these options from the drop down:

- SAM-Account-Name:** Common Name
- Display-Name:** Name
- E-Mail-Addresses:** E-mail Address
- SAM-Account-Name:** Name ID

- Add a rule to send group membership as a claim:
- Claim rule template:** Send Group Membership as a Claim
- Claim rule name:** Group membership- <GROUP NAME>
- User's group:** <DOMAIN NAME>\<GROUP NAME>
- Outgoing claim type:** Group
- Outgoing claim value:** <GROUP NAME>
- Add additional rules for passing group membership of other ADFS user groups at your organization.

Configure an RBAC group and role in PE

In the PE console, configure RBAC to grant permissions to new ADFS user groups.

- In the console, on the **Access control** page, click the **User groups** tab.

- In the **Login** field, enter the name of the ADFS user group and click **Add Group**.

Tip: This is the same <GROUP NAME> you added when configuring group membership rules.

- Click the **User roles** tab, then click the role you want to add the group to. For example, **Viewers**.
- Click the **Member groups** tab and, in the drop-down list, select your ADFS user group.
- Click **Add group** and commit the change.
- Add additional ADFS user groups at your organization to RBAC roles.

Test your SSO connection

Ensure your connection between PE and ADFS works by logging out and logging back in.

- Log out of PE.
- On the login screen, click **Sign in with ADFS**.
- Log in to PE using your ADFS credentials.

After logging back in, your permissions match what is assigned to your ADFS group.

Connect Okta to PE

Connect to Puppet Enterprise (PE) to Okta so that users can log in to PE with their Okta credentials.

These steps assume you're familiar with common SAML terminology and the basic process to [Connect a SAML identity provider to PE](#) on page 289.

You must have an Okta instance. To test this process, you might request a development instance from the [Okta Developer Portal](#).

Configure the Okta application

Configure settings in Okta to connect your Okta instance to Puppet Enterprise (PE).

Before you begin

[Get URLs and the signing and encryption certificate](#) on page 289 required to connect Okta to PE.

- Log in to the Okta Admin Console and navigate to **Applications > Applications > Create App Integration**. The **App Integration Wizard** starts.
- Select **SAML 2.0** for the **Sign-in method**, and click **Next**.
- On the **General Settings** tab:
 - Enter **Puppet Enterprise** for the **App name**.
 - Optional: Upload an **App logo** and select **App visibility** options.
 - Click **Next**.
- On the **Configure SAML** tab:
 - Paste the SAML assertion consumer service (ACS) URL from PE in the **Single sign on URL** field.
 - Paste the SAML metadata URL from PE in the **Audience URI (SP Entity ID)** field.
 - Optional: Set the **Default RelayState**.
 - Select a **Name ID format** and **Application username**.

5. Click **Advanced Settings**, and specify parameters that you'll match to service provider configuration options in PE later.
 - a) Select options for **Response**, **Assertion Signature**, **Signature Algorithm**, **Digest Algorithm**, and **Assertion Encryption**.
 - b) Select **Allow application to initiate Single Logout**, and then paste the SAML Single Logout URL from PE in the **Single Logout URL** field.
 - c) Paste the SAML assertion consumer service (ACS) URL from PE in the **SP Issuer** field.
 - d) For the **Signature Certificate**, upload the file containing the Signing and Encryption Certificate from PE.
 - e) Configure the **Assertion Inline Hook**, **Authentication context class**, **Honor Force Authentication**, and **SAML Issuer ID**.

Tip: Take note of the **Authentication context class** setting. You'll need this value when you configure the Okta connection settings in PE.

6. Click **Next**, complete the feedback survey (if desired), and then click **Finish**.
7. Copy the URLs and download the certificate from the [How to Configure SAML 2.0 for Puppet Enterprise Application](#) page. You'll need this information to connect to Okta in the PE console.

[Connect to Okta in the PE console](#) on page 301

Connect to Okta in the PE console

Configure your Okta integration settings in the Puppet Enterprise (PE) console.

Before you begin

You need the URLs and certificate from the [How to Configure SAML 2.0 for Puppet Enterprise Application](#) page (which appears after you [Configure the Okta application](#) on page 300). You also need to know the values of the **Signature Algorithm** and **Authentication context class** settings in Okta.

For more information about PE's SAML configuration fields and their corresponding IdP and RBAC API mappings, refer to the [SAML configuration reference](#) on page 291 and Okta's documentation.

1. In the console, on the **Access control** page, click the **SSO** tab.
2. Click **Configure**.
3. Input a **Display Name**. This name is visible on the PE home page.
4. Complete the **Identity provider information** fields:
 - **Identity provider entity ID**: Input the **Identity Provider Issuer URL** from Okta.
 - **Identity provider SSO URL**: Input the **Identity Provider Single Sign-On URL** from Okta.
 - **Identity provider SLO URL**: Input the **Identity Provider Single Logout URL** from Okta.
 - **Identity provider SSO response URL**: Optional and can be blank.
 - **Identity provider certificate**: Paste the entire **X.509 Certificate** from Okta, including the begin and end tags.

5. Configure the **Service provider configuration options** as follows:

- **Name ID encrypted?**: Yes
- **Sign authentication requests?**: Yes
- **Sign logout response?**: Yes
- **Sign logout requests?**: Yes
- **Require signed messages?**: Yes
- **Require signed assertions?**: Yes
- **Sign metadata?**: Yes
- **Require encrypted assertions?**: No (leave unselected)
- **Require name ID encryption?**: No (leave unselected)
- **Requested authentication context**: Input the value of the **Authentication context class** from Okta in the following format:

```
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
```

- **Requested authentication context comparison**: Select **minimum**
- **Allow duplicated attribute name**: No (leave unselected)
- **Validate xml?**: No (leave unselected)
- **Signature algorithm**: Must match the **Signature Algorithm** setting you chose in Okta, such as **rsa-sha256**

6. Input **Organization** and **Contacts** information.

7. The values in the **Attribute binding** fields must exactly match the corresponding fields in Okta.

These settings define attributes and map them to user information in Okta, then PE uses these settings to understand user information received from Okta.

Your Okta Administrator can provide these details, or you can retrieve them from Okta. Navigate to **Applications** > **SAML General** > **Advanced settings** > **Attribute Statements**, and then use the values from the **Name** fields in Okta to populate the **Attribute binding** fields in PE.

8. Commit your changes.

[Configure RBAC for an Okta integration](#) on page 302

Configure RBAC for an Okta integration

In the PE console, connect Okta user groups to PE RBAC roles.

1. In the console, on the **Access control** page, click the **User roles** tab.
2. Click the **Name** of the PE role you want to connect to an Okta user group.
3. On the **Member users** tab, select the Okta data from the **User name** drop-down menu, such as `$(user.firstName) $(user.lastName)`.

The value for this option derives from the **Attribute Statements** data in Okta. If no such value is available on the drop-down menu, check the **Attribute binding** settings in PE (refer to [Connect to Okta in the PE console](#) on page 301 for details).

The **Login** and **Status** fields automatically populate after you select the **User name**.

4. Switch to the **Member groups** tab and select the relevant Okta group from the **Group name** drop-down menu.
5. Commit the changes.
6. Repeat to configure additional groups.

Test your Okta SSO connection

Make sure you can log in to PE with Okta.

1. Log out of PE.
2. Go to the PE login screen (home page) and click **Sign in with Okta SSO**.
3. Log in to PE using your Okta credentials.

If the configuration is correct, you'll be redirected to the PE status page. Make sure you have the correct permissions.

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

Authentication tokens manage access to these Puppet Enterprise (PE) services:

- Activity service
- Code Manager
- Node classifier
- PuppetDB
- Puppet orchestrator
- RBAC

You can generate authentication tokens using the PE console, the `puppet-access` command, or the RBAC API v1 [Tokens endpoints](#) on page 342. You can also generate one-off tokens that do not need to be saved, which are typically used by a service.

In the PE console, you can view or revoke your own tokens on the **Tokens** tab of the **My account** page. Administrators can view and revoke tokens for other users on the **User details** page. You can also [Configure RBAC and token-based authentication settings](#) on page 224 in the **PE Infrastructure** node group.

Related information

[Installing client tools](#) on page 172

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.

[Reference: User permissions and names](#) on page 270

This reference describes the permissions granted to the five default Puppet Enterprise (PE) user roles, as well as the *display name* and *system name* for each type and permission.

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 define default settings so that you can 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 (PE), the global configuration file is created for you. The configuration file is formatted in JSON. For example:

```
{
  "service-url": "https://<CONSOLEROHOSTNAME>:4433/rbac-api",
  "token-file": "~/.puppetlabs/token",
  "certificate-file": "/etc/puppetlabs/puppet/ssl/certs/ca.pem"
```

```
}
```

Tip: PE determines and populates 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 required 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.

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 setting prevails.

You must create the user-specified file and populate it with the configuration file settings. A list of configuration file settings is found in [Configuration file settings for puppet-access](#) on page 304.

Important: User-specified configuration files must be in JSON format. HOCON and INI-style formatting are not supported.

Configuration file settings for `puppet-access`

You can manually add or edit configuration settings in your user-specified or global `puppet-access` configuration files.

The class that manages the global configuration file is: `puppet_enterprise::profile::controller`

You can also change configuration settings by specifying flags when you [Generate a token using puppet-access](#) on page 305 on 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</code> or <code>--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</code> or <code>--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, which is usually: <code>https://<CONSOLE_HOSTNAME>:4433/rbac-api</code> Usually, you need to change this only if you are moving your console server.	<code>--service-url</code>

Generate a token in the console

Use the console to generate an authentication token that you can use to access PE APIs. If SAML is configured, you must have a token to use CLI tools, such as 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 the `puppet-access` command 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` on page 303.

For information about modifying commands for Windows and privilege escalation, refer to Using example commands on page 28 and Commands with elevated privileges on page 30.

1. Choose one of the following options, depending on how long you need your token to last:

- To generate a token with the default one-hour lifetime, run:

```
sudo puppet-access login
```

- To generate a token with a specific lifetime, run:

```
sudo puppet-access login --lifetime <TIME_PERIOD>
```

For example, to generate a token that lasts five hours, run:

```
puppet-access login --lifetime 5h
```

2. When prompted, enter the user name and password that you use to log into the PE console.

The `puppet-access` command uses RBAC API v1 [Tokens endpoints](#) on page 342. If your login credentials are correct, the RBAC service generates a token.

The token is generated and stored in a file for later use. The default token storage location is `~/ .puppetlabs/ token`. You can print the token at any time, such as in curl commands, by using `puppet-access show`.

You can continue to use this token until it expires, or until your access is revoked. The token has the same 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.

Important: If a remote user generates a token, and the user is then 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 308

If you want a token to have a different lifetime than the default lifetime, you can set a different lifetime when you generate the token. This allows you to keep one token for multiple sessions.

Generate a token using the RBAC API

The RBAC API v1 /auth/token endpoint allows you to generate a token.

1. Call the [POST /auth/token](#) on page 343 or [POST /tokens](#) on page 344 endpoint.

2. Save the token by:

- Copying the token to a text file.
- Saving the token as an environment variable using: `export TOKEN=<TOKEN>`

You can use the token until it expires, or until your access is revoked. The token has the same permissions as the user associated with it.

Important: If a remote user generates a token, and that user is then deleted from your external directory service, the deleted user cannot log into the Puppet Enterprise (PE) 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 prevent the user from accessing the system through the token, you need to manually [revoke](#) or [delete](#) the user from PE.

Use a token with PE API endpoints

The example below shows how to use a token in an API endpoint request. For more information, refer to the documentation for the particular API or endpoint you want to use.

Before you begin

Generate a token using `puppet-access login`.

1. [Generate a token in the console](#) on page 291, [Generate a token using puppet-access](#) on page 305, or [Generate a token using the RBAC API](#) on page 306.

2. Supply the token in the endpoint using one of these methods (if necessary, replacing /etc/puppetlabs/puppet/ssl/certs/ca.pem with the correct path to your CA certificate file):
 - An X-Authentication header using `puppet-access show` to call a stored token:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current"

curl --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" --header
"$auth_header" "$uri"
```

- An X-Authentication header with the token supplied in full:

```
auth_header="X-Authentication: <TOKEN>"
uri="https://$(puppet config print server):4433/rbac-api/v1/users/current

curl --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" --header
"$auth_header" "$uri"
```

- Appended as a query parameter:

```
GET https://$(puppet config print server):4433/rbac-api/v1/users/current?token=<TOKEN>"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Generate a token for use by a service

If you need to generate a token that a Puppet Enterprise (PE) service can use, and the token doesn't need to be saved, use the `--print` option with the `puppet-access` command.

Before you begin

[Install the PE client tools package](#) and [Configure puppet-access](#) on page 303.

To generate a token for a service, run:

```
sudo puppet-access login [username] --print
```

This command generates a token, and then displays the token content as stdout (standard output) rather than saving it to disk.

Tip: When generating a token for a service, consider specifying a longer [token lifetime](#) so that you don't have to regenerate the token too frequently.

For information about modifying commands for Windows and privilege escalation, refer to [Using example commands](#) on page 28 and [Commands with elevated privileges](#) on page 30.

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 default token lifetime

Tokens have a default authentication lifetime of one hour, but this default value can be adjusted in the console. You can also change the maximum permitted 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 the parameter you want to adjust:
 - `rbac_token_auth_lifetime`: Set the default token lifetime. The default is one hour.
 - `rbac_token_maximum_lifetime`: Set the maximum allowable lifetime for all tokens. The default is 10 years.
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)
- `s` (seconds)

For example, `12h` sets the lifetime to 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 (`s`).

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

If you want a token to have a different lifetime than the default lifetime, you can set a different lifetime when you generate the token. This allows you to keep one token for multiple sessions.

If you [Generate a token using puppet-access](#) on page 305, use the `--lifetime` option. For example: `puppet-access login --lifetime 2h` generates a token with a two-hour lifetime.

If you're using the [POST /auth/token](#) on page 343 endpoint, use the `lifetime` key. For example, this JSON body specifies a token lifetime of two hours:

```
{ "login": "<YOUR PE USER NAME>" , "password": "<YOUR PE PASSWORD>" ,
  "lifetime": "2h" }
```

Format the lifetime as a numeric value followed by one of the following:

- `y` (years)
- `d` (days)
- `h` (hours)
- `m` (minutes)
- `s` (seconds)

For example, `12h` sets the lifetime to 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 (`s`).

To set the maximum possible lifetime, set the lifetime to 0. This sets the lifetime to the value of `rbac_token_maximum_lifetime`. The default value for this setting is 10 years.

If omitted, tokens get the default lifetime, which is one hour, unless you [Change the default token lifetime on page 308](#).

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 help you quickly call a token when working with RBAC API endpoints or when revoking your own token.

To generate a token with a label, use the `label` key in requests to the [POST /auth/token](#) on page 343 endpoint. The value of the `label` key becomes the token's label. For example:

```
{ "login": "<YOUR_PE_USER_NAME>" ,
  "password": "<YOUR_PE_PASSWORD>" ,
  "label": "My token" }
```

Labels:

- Can't have more than 200 characters.
- Can't contain commas.
- Can't contain only spaces.

Whitespace is allowed within the label string; however, leading and trailing whitespace is trimmed. For example, "my token label" becomes "my token label".

Token labels are assigned on a per-user basis. This means two users can both 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.

Revoke a token using the API

Revoke tokens by username, label, or full token with the [DELETE /tokens](#) on page 364 endpoint.

All token revocation attempts are logged in the activity service, and they 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 access can also revoke those users' tokens, because the `users:disable` permission includes token revocation. Revoking users' tokens does not [revoke](#) the users' PE accounts. 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 other users' tokens.

Administrators can revoke another user's token on the **User details** page.

To revoke your own 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**.

Delete a token file

If you used `puppet-access` to generate a token, you can remove the token file by running the `delete-token-file` action. This is useful if you are working on a server that is used by multiple people.

Deleting the token file prevents other users from using your authentication token, but does not 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 <TOKEN_PATH>
```

RBAC API

Use the RBAC API to manage users, user groups, roles, permissions, tokens, password, and LDAP or SAML connections.

Endpoint	Use
users	Manage local users as well as those from a directory service, get lists of users, and create new local users. This endpoint has a v1 and v2. The v2 GET /users on page 360 endpoint has more filtering options.
groups	Get lists of groups and add a new remote user group. This endpoint has a v1 and v2. The v2 POST /groups on page 362 endpoint has the option to validate the group against LDAP before creating it.
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. This endpoint has a v1 and v2. Use the v2 GET /ds on page 367 endpoint to get information about your directory service.
saml	Configure SAML, get SAML configuration details, and get the public certificate and URLs for configuration.
password	Generate password reset tokens and update user passwords.
tokens	Generate authentication tokens to access PE. Use the v1 token endpoints to create tokens, and use the v2 token endpoints to revoke and validate tokens.
rbac-service	Use the Status API to check the status of the RBAC service.

- [Forming RBAC API requests](#) on page 311

The role-based access control (RBAC) API accepts well-formed HTTPS requests. Token-based authentication is required for most endpoints. You can use either user authentication tokens or allowed certificates to authenticate requests.

- [RBAC service errors](#) on page 313

RBAC API error responses can be formatted as `text/html` or JSON objects.

- [RBAC API v1](#) on page 316

Use the role-based access control (RBAC) API v1 endpoints to manage users, directory service groups, roles, permissions, tokens, passwords, and LDAP and SAML connection settings.

- [RBAC API v2](#) on page 359

The role-based access control (RBAC) API v2 service enables you to fetch information about users, create groups, revoke tokens, validate tokens, and get information about your LDAP directory service.

Related information

[Status API](#) on page 416

You can use the status API to check the health of Puppet Enterprise (PE) components and services. It is useful for automatically monitoring your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

Forming RBAC API requests

The role-based access control (RBAC) API accepts well-formed HTTPS requests. Token-based authentication is required for most endpoints. You can use either user authentication tokens or allowed certificates to authenticate requests.

RBAC API requests must include a URI path following the pattern:

```
https://<DNS>:4433/rbac-api/<VERSION>/<ENDPOINT>
```

The variable path components derive from:

- **DNS:** Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).
- **VERSION:** Either v1 or v2, depending on the endpoint.
- **ENDPOINT:** One or more sections specifying the endpoint, such as `users` or `roles`. Some endpoints require multiple sections, such as the [POST /command/roles/add-users](#) on page 334 endpoint.

Tip: The RBAC service listens on port 4433 by default. If you change the RBAC service's port, you'll need to change the port in your API calls.

For example, you could use any of these paths to call the [GET /users](#) on page 317 endpoint:

```
https://$(puppet config print server):4433/rbac-api/v1/users
https://localhost:4433/rbac-api/v1/users
https://puppet.example.dns:4433/rbac-api/v1/users
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Token authentication

For most RBAC API endpoints, you must authenticate your requests with user authentication tokens. For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

To use a token in an RBAC API request, you can use `puppet-access show`, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/rbac-api/v1/users"

curl --header "$auth_header" "$uri"
```

Or you can use the actual token, such as:

```
auth_header="X-Authentication: <TOKEN>
uri="https://$(puppet config print server):4433/rbac-api/v1/users"

curl --header "$auth_header" "$uri"
```

For some endpoints, you might append the token (or token variable) directly to the URI path, such as:

```
https://$(puppet config print server):4433/rbac-api/v1/users/current?token=
$(puppet-access show)
https://$(puppet config print server):4433/rbac-api/v1/users/current?
token=<TOKEN>
```



CAUTION: Tokens supplied as query parameters might be recorded in server access logs.

Allowed certificate authentication

You can authenticate requests with a certificate listed in RBAC's certificate allowlist, which is located at:

```
/etc/puppetlabs/console-services/rbac-certificate-allowlist
```

Important: If you edit the `rbcac-certificate-allowlist` file, you must reload the `pe-console-services` service for your changes to take effect. To reload the service run: `sudo service pe-console-services reload`

To use a certificate in a curl command, include the allowed certificate name (which must match a name in the `rbcac-certificate-allowlist` file) and, if necessary, the private key. For example:

```
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"
```

Tip: You do not need to use an agent certificate for authentication. You can use the `puppet cert generate` command to create a certificate to use specifically with the RBAC API.

Permissions objects

Payloads that use JSON objects for permissions must represent each of the three components: Type (`object_type`), permission (`action`), and object (`instance`), as described in [Structure of user permissions](#) on page 269. In RBAC API requests, you must use the system names (not the display names) described in [Reference: User permissions and names](#) on page 270.

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

RBAC API Content-Type headers

The RBAC API accepts only JSON payloads in PUT and POST requests.

If you provide a JSON payload, you must specify that the content is in JSON format. Thus, all PUT and POST requests with non-empty bodies must have the Content-Type header set to application/json.

RBAC service errors

RBAC API error responses can be formatted as text/html or JSON objects.

Error response format

RBAC API error responses can use the following keys:

Key	Definition
kind	The kind of error encountered.
msg	A human-readable message associated with the error.
details	For error responses formatted as text/html, the body is the contents of this key. Additional, potentially machine-readable, information about the error condition.

General error responses

RBAC API endpoints that accept a JSON body might return these responses.

Response	Response code	Description
malformed-request	400	The submitted data is not valid JSON. The details key contains an error message from the JSON parser.

Response	Response code	Description
schema-violation	400	<p>The submitted data has an unexpected structure, such as invalid fields or missing required fields. The <code>msg</code> key describes the problem, and the <code>details</code> key is an object containing:</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	ID data in the request body doesn't match the ID in the request's URI path. The <code>details</code> key shows the two IDs.
invalid-id-filter	400	The request's URI path 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 an endpoint that requires authentication.
user-revoked	401	A revoked user attempted to access an endpoint that requires authentication.
api-user-login	401	A person attempted to log in as the <code>api_user</code> with a password. The <code>api_user</code> does not support username/password authentication.

Response	Response code	Description
remote-user-conflict	401	A remote user who is not yet known to RBAC attempted to authenticate, but a local user with the same login already exists. The solution is to change either the local user's login in RBAC, or to change the remote user's login. To change the remote user's login you can either change the <code>user_lookup_attr</code> in the DS settings or change the value in the directory service itself.
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 user roles 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	You submitted a value for a field that is supposed to be unique, but another object already has that value. For example, when you attempt to create a user 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 LDAP user or group IDs, and one or more of those IDs does not correspond to an existing LDAP user or group.
no-such-group-LDAP		
non-unique-lookup-attr	422	A login was attempted, but LDAP found multiple users with the given username. Your 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 are usually available in the logs.

RBAC API v1

Use the role-based access control (RBAC) API v1 endpoints to manage users, directory service groups, roles, permissions, tokens, passwords, and LDAP and SAML connection settings.

- [Users endpoints](#) on page 316

With role-based access control (RBAC), you can manage local users and remote users (created on a directory service). Use the `users` endpoints to get lists of users, create local users, and delete, revoke, and reinstate users in PE.

- [User groups endpoints](#) on page 326

User groups allow you to quickly assign one or more roles to a set of users by placing all relevant users in the group. This is more efficient than assigning roles to each user individually. Use the `groups` endpoints to get lists of groups and add, delete, and change groups.

- [User roles endpoints](#) on page 330

User roles contain sets of permissions. When you assign a user (or a user group) to a role, you can assign the entire set of permissions at once. This is more organized and easier to manage than assigning individual permissions to individual users. Use the `roles` endpoints to manage roles.

- [Permissions endpoints](#) on page 338

You add permissions to roles to control what users can access and do in PE. Use the `permissions` endpoints to get information about objects you can create permissions for, what types of permissions you can create, and whether specific users can perform certain actions.

- [Tokens endpoints](#) on page 342

Authentication tokens control access to PE services. Use the `auth/token` and `tokens` endpoints to create tokens.

- [LDAP endpoints](#) on page 345

Use the LDAP `ds` (directory service) endpoints to get information about your LDAP directory service, test your LDAP directory service connection, and replace LDAP directory service connection settings.

- [SAML endpoints](#) on page 350

Use the `saml` endpoints to configure SAML, retrieve SAML configuration details, and get the public certificate and URLs needed for configuration.

- [Passwords endpoints](#) on page 353

When local users forget their Puppet Enterprise (PE) 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. Use the `password` endpoints to generate password reset tokens, use tokens to reset passwords, change the authenticated user's password, and validate potential user names and passwords.

- [Disclaimer endpoints](#) on page 357

Use these endpoints to modify the disclaimer text that appears on the Puppet Enterprise (PE) console login page.

Related information

[API index](#) on page 33

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

Users endpoints

With role-based access control (RBAC), you can manage local users and remote users (created on a directory service). Use the `users` endpoints to get lists of users, create local users, and delete, revoke, and reinstate users in PE.

You can:

- [GET /users](#) on page 317: Get a list of all local and remote users.
- [GET /users/<sid>](#) on page 319: Get information about specific users.
- [GET /users/current](#) on page 320: Get information about the current authenticated user.
- [GET /users/<sid>/tokens](#) on page 320: Get a list of tokens for a user.
- [POST /users](#) on page 321: Create a local user.
- [PUT /users/<sid>](#) on page 322: Edit a user.
- [DELETE /users/<sid>](#) on page 323: Delete a user from the PE console.
- [POST /command/users/add-roles](#) on page 324: Assign roles to a user.

- [POST /command/users/remove-roles](#) on page 325: Remove roles from a user.
- [POST /command/users/revoke](#) on page 325: Revoke a user's PE access.
- [POST /command/users/reinstate](#) on page 325: Reinstate a revoked user.

Tip: You'll want to be familiar with the [Users endpoints keys](#) on page 317 that appear in these endpoints' requests and responses.

Users endpoints keys

These keys are used with the RBAC API v1 `users` endpoints.

Key	Definition	Example
<code>id</code>	A UUID string identifying the user.	"4f ee7450-54c7-11e4-916c-0800200c9a
<code>login</code>	A string used by the user to log in. Must be unique among users and groups.	"admin"
<code>email</code>	An email address string. Not currently utilized by any code in PE.	"hill@example.com"
<code>display_name</code>	The user's name as a string.	"Kalo Hill"
<code>role_ids</code>	An array of role IDs indicating roles to directly assign to the user. An empty array is valid.	[3 6 5]
<code>is_group</code>	These flags indicate whether a user is remote and/or a super user. For all users, <code>is_group</code> is always <code>false</code> .	true or false
<code>is_remote</code>		
<code>is_superuser</code>		
<code>is_revoked</code>	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 or false
<code>last_login</code>	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"
<code>inherited_role_ids</code> (remote users only)	An array of role IDs indicating which roles a remote user inherits from their groups.	[9 1 3]
<code>group_ids</code> (remote users only)	An array of UUIDs indicating which groups a remote user inherits roles from.	["3a96d280-54c9-11e4-916c-0800200c9a

GET /users

Fetches all local and remote users, including the superuser. You can also query specific users by user ID. Authentication is required.

Important: You can use the v1 GET `/users` endpoint to query all users or specific users by ID; however, the v2 [GET /users](#) on page 360 endpoint provides more query options and control over the response content.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users" -H "X-Authentication:$(puppet-access show)"
```

To query specific users, append a comma-separated list of user IDs:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users?id=<SID>,<SID>" -H "X-Authentication:$(puppet-access show)"
```

Response format

The response is a JSON object that contains 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"
}]
```

For information about keys in the response, refer to [Users endpoints keys](#) on page 317.

For information about error responses, refer to [RBAC service errors](#) on page 313.

GET /users/<sid>

Fetches information about specific users identified by subject ID (<sid>). Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, provide authentication and specify a user ID, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users/<SID>" -H
  "X-Authentication:$(puppet-access show)"
```

To request multiple users, append a comma-separated list of user IDs, such as

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users?
  id=<SID>,<SID>" -H "X-Authentication:$(puppet-access show)"
```

Tip: Querying multiple users technically calls the [GET /users](#) on page 317 endpoint with the `id` parameter.

Response format

The response is a JSON object that contains metadata for the requested user (or users). For example, this response is for a local user:

```
{"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"}
```

And this response is for a remote user:

```
{"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"}
```

For information about keys in the response, refer to [Users endpoints keys](#) on page 317.

For information about error responses, refer to [RBAC service errors](#) on page 313.

GET /users/current

Fetches data about the current authenticated user. The user's ID is assumed from the authentication context. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users/current"
-H "X-Authentication:$(puppet-access show)"
```

Response format

The response is the same as [GET /users/<sid>](#) on page 319.

GET /users/<sid>/tokens

Fetches a list of tokens for a given user. Authentication required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, you must supply a user ID in the URI path, such as:

```
https://$(puppet config print server):4433/rbac-api/v1/users/
c97c716a-5f42-49d8-b5a4-d0888a879d21/tokens
```

You can append these optional parameters to the request:

Parameter	Definition
limit	An integer specifying the maximum number of records to return. If omitted, all records are returned.
offset	Specify a zero-indexed integer to specify the index value of the first record to return. If omitted, the default is position 0 (the first record). For example, <code>offset=5</code> would start from the 6th record.
order_by	Specify one of the following strings to define the order in which records are returned: <ul style="list-style-type: none"> • <code>creation_date</code> • <code>expiration_date</code> • <code>last_active_date</code> • <code>client</code> If omitted, the default is <code>creation_date</code> .
order	Determines the sort order as either ascending (<code>asc</code>) or descending (<code>desc</code>). If omitted, the default is <code>asc</code> .

For example:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/users/<SID>/tokens?limit=20" \
-H "X-Authentication:$(puppet-access show)"
```

Response format

The response is a JSON object describing each token and the pagination information from the request.

Tokens are containing in an `items` array. Each token is represented as an object using these keys:

- `id`: The token's ID
- `creation_date`: The date and time the token was created in ISO-8601 format.
- `expiration_date`: The date and time the token expires (or expired) in ISO-8601 format.
- `last_active_date`: The date and time the token was last used in ISO-8601 format.
- `client`: Client information.
- `description`: Arbitrary description information.
- `session_timeout`: An integer, present with a timeout (in minutes), if this is a session-based token.
- `label`: A label, if one was supplied at creation. Refer to [Set a token-specific label](#) on page 309.

The `pagination` object reiterates the query parameters from the request as well as the total number of records available (regardless of `limit` or `offset`).

For example:

```
{"items": [ {
    "id": <token_id>
    "creation_date": <ISO-8601>,
    "expiration_date": <ISO-8601>,
    "last_active_date": <ISO-8601>,
    "client": "",
    "description": "",
    "session_timeout": ,
    "label": ""
  }, ...
],
"pagination": {
    "limit": 20,
    "offset": 0,
    "order_by": "creation_date",
    "order": "asc"
    "total": 25
  }
}
```

Error response

If a user with the provided use ID doesn't exist, the endpoint returns a `404 Not Found` response.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /users

Create a local user. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `email`: Specify the user's email address.

- `display_name`: The user's name as you want it shown in the console.
- `login`: The username for the user to use to login.
- `role_ids`: An array of role IDs defining the roles that you want to assign to the new user. An empty array is valid, but the user can't do anything in PE if they are not assigned to any roles.
- `password`: A password the user can use to login. For the password to work in the PE console, it must be at least six characters. This field is optional, however user accounts are not usable until a password is set. You can also use the [Passwords endpoints](#) on page 353 to generate a password reset token the user can use to login for the first time.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/users"
  \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"login": "Kalo",
  "email": "kalohill@example.com",
  "display_name": "Kalo Hill",
  "role_ids": [1123,6643,1218],
  "password": "Welc0me!"}
```

Response format

If creation is successful, the endpoint returns 201 `Created` with a location header pointing to the new resource.

Error responses

If the email or login for the user conflicts with an existing user's login, the endpoint returns a 409 `Conflict` response.

For other errors, refer to [RBAC service errors](#) on page 313.

PUT /users/<sid>

Replace the content of the specified user object. For example, you can update a user's email address or role assignments. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using all keys supplied in the [GET /users/<sid>](#) on page 319 endpoint response, modified as needed to update the user. For descriptions of each key, refer to [Users endpoints keys](#) on page 317. Not all keys can be updated, such as `last_login`.

The `role_ids` array indicates the roles to assign to the user. An empty `role_ids` array removes all roles directly assigned to the user.

An example, this JSON body is for a local user:

```
{"id": "c8b2c380-5889-11e4-8ed6-0800200c9a66",
"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"}
```

And this body is for a remote user:

```
{
  "id": "3271fde0-588a-11e4-8ed6-0800200c9a66",
  "login": "Jean",
  "email": "jeanjackson@example.com",
  "display_name": "Jean Jackson",
  "role_ids": [4, 1],
  "inherited_role_ids": [],
  "group_ids": [],
  "is_group": false,
  "is_remote": true,
  "is_superuser": false,
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z"
}
```

Here is an example of a complete curl request to this endpoint:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/users/c97c716a-5f42-49d8-b5a4-d0888a879d21" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
  "id": "c97c716a-5f42-49d8-b5a4-d0888a879d21",
  "login": "replace-test",
  "email": "replace-test@example.com",
  "display_name": "Replaced User",
  "role_ids": [],
  "is_group": false,
  "is_remote": false,
  "is_superuser": false,
  "is_revoked": false,
  "last_login": "2014-05-04T02:32:00Z"
}'
```

Response format

Returns 200 OK and a user object showing the changes made. For example:

```
{
  "email": "replace-test@example.com",
  "is_revoked": false,
  "last_login": null,
  "is_remote": false,
  "login": "replace-test",
  "is_superuser": false,
  "id": "c97c716a-5f42-49d8-b5a4-d0888a879d21",
  "role_ids": [],
  "display_name": "Replaced User",
  "is_group": false
}
```

Error responses

If the user's login user conflicts with an existing user login, the endpoint returns a 409 Conflict response.

For other errors, refer to [RBAC service errors](#) on page 313.

DELETE /users/<sid>

Deletes the user with the specified ID, regardless of whether they are a user defined in PE RBAC (local) or a user defined by a directory service (remote). Authentication is required.

Remember: The admin user and the api_user can't be deleted. The API user is for service-to-service authentication within PE. It cannot be used with the standard login, and it is only available through certificate-based authentication. The RBAC *allow list* identifies (by certname) the certificates you can use for API user authentication.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the URI path must include the ID of the user you want to delete from the PE console. For example:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v1/
users/76351f96-3d89-4947-bde9-bc3d86542839" \
-H "X-Authentication:$(puppet-access show)"
```

Response format

If the user is successfully deleted, the endpoint returns a 204 No Content response.

Important: When removing directory service users (remote users), this action removes the user from the PE console, but the user is still able to log in if they are not revoked. When a non-revoked directory service user logs in, their account is re-added to the console. Make sure to use the [POST /command/users/revoke](#) on page 325 endpoint to revoke the user's access.

Error responses

If the requesting user (based on the authentication in the request) does not have the `users:edit` permission for the specified user, the endpoint returns a 403 Forbidden response.

If there is no user with the provided ID, the endpoint returns a 404 Not Found response.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/users/add-roles

Assign roles to a user.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `user_id`: The ID of the user you want to assign roles to.
- `role_ids`: An array of role IDs defining the roles that you want to assign to the user. An empty array is valid, but the user can't do anything in PE if they are not assigned to any roles.

Example payload:

```
{ "user_id": <user-id>, "role_ids": [1,2,3] }
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/users/add-roles" \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{ "user_id": "c97c716a-5f42-49d8-b5a4-d0888a879d21", "role_ids": [1] }'
```

Tip: To assign multiple users to the same role at once, use the [POST /command/roles/add-users](#) on page 334 endpoint.

Response format

Returns 204 No Content if the roles are successfully assigned to the user.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/users/remove-roles

Remove roles from a user.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `user_id`: The ID of the user you want to remove roles from.
- `role_ids`: An array of role IDs defining the roles that you want to remove from the user.

Example payload:

```
{ "user_id": <user-id>, "role_ids": [1,2,3] }
```

Example curl request:

```
curl -X POST 'https://$(puppet config print server):4433/rbac-api/v1/
command/users/remove-roles' \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"user_id": "c97c716a-5f42-49d8-b5a4-d0888a879d21", "role_ids": [1]}
```

Response format

Returns 204 No Content if the user exists and the roles are successfully unassigned.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/users/revoke

Revoke a user's access to PE.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object containing the `user_id` key, which specifies the ID of the user you want to revoke.

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/users/revoke" \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"user_id": "c97c716a-5f42-49d8-b5a4-d0888a879d21"}'
```

Response format

Returns 204 No Content if the user is revoked successfully.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/users/reinstate

Reinstate a user after they have been revoked.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object containing the `user_id` key, which specifies the UUID of the user you want to reinstate.

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/users/reinstate" \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"user_id": "c97c716a-5f42-49d8-b5a4-d0888a879d21"}'
```

Response format

Returns 204 No Content if the user is successfully reinstated.

Returns 404 Not Found if the specified user doesn't exist.

For other errors, refer to [RBAC service errors](#) on page 313.

User groups endpoints

User groups allow you to quickly assign one or more roles to a set of users by placing all relevant users in the group. This is more efficient than assigning roles to each user individually. Use the groups endpoints to get lists of groups and add, delete, and change groups.

Remember: Group membership is determined by your directory service hierarchy. Therefore, local users (that exist only in the PE console) can't be in directory groups. You'll need to use the [Users endpoints](#) on page 316 to manage these users' roles.

User groups endpoints keys

These keys are used with the RBAC API v1 groups endpoints.

Key	Definition	Example
id	A UUID string identifying the group.	"c099d420-5557-11e4-916c-0800200c9a
login	The identifier for the user group on the directory server.	"admins"
display_name	The group's name as a string.	"Admins"
role_ids	An array of role IDs indicating roles to assign to the group's members. An empty array is valid.	[3 6 5]
Tip: This is the only field that can be updated via RBAC; the rest are immutable or synced from the directory service.		
is_group	These flags indicate that the group is a group, derived from the directory service, and not a super user (inherently, a group can't be a user).	These are set to true, true, and false, respectively
is_remote		
is_superuser		
is_revoked	No effect. Because groups are not user objects, setting this flag to true does nothing.	true or false
user_ids	An array of UUIDs indicating which users belong to the group.	["3a96d280-54c9-11e4-916c-0800200c9a

GET /groups

Fetch information about all user groups. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/groups" \
-H "X-Authentication:$(puppet-access show)"
```

To query specific groups, append a comma-separated list of group IDs:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/groups?
id=<SID>,<SID>" \
-H "X-Authentication:$(puppet-access show)"
```

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": "admins",
  "display_name": "Admins",
  "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": "owners",
  "display_name": "Owners",
  "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": "ccdbdbe50-588a-11e4-8ed6-0800200c9a66",
  "login": "viewers",
  "display_name": "Viewers",
  "role_ids": [2, 3],
  "is_group": true,
  "is_remote": true,
  "is_superuser": false,
  "user_ids": []
} ]
```

For information about keys in the response, refer to [User groups endpoints keys](#) on page 326.

For information about error responses, refer to [RBAC service errors](#) on page 313.

GET /groups/<sid>

Fetches information about a single user group identified by ID. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, provide authentication and specify a group ID, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/groups/<SID>" \
-H "X-Authentication:$(puppet-access show)"
```

To request information for multiple specific groups, use the [GET /groups](#) on page 327 endpoint with the `id` parameter.

Response format

The response is a JSON object containing information about the requested 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"]}
```

For information about keys in the response, refer to [User groups endpoints keys](#) on page 326.

Error responses

If the user who submits the GET request has not successfully authenticated, the endpoint returns a 401 Unauthorized response.

If the requesting user does not have the appropriate user permissions to request the group data, the endpoint returns a 403 Forbidden response.

For other error responses, refer to [RBAC service errors](#) on page 313.

POST /groups

Creates a new remote directory user group. Authentication is required.

Tip: The v2 [POST /groups](#) on page 362 endpoint supports more options for creating groups, including assigning a display name and the option to validate if the group exists on the LDAP server before creating it.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `login`: Defines the group for an external IdP. This could be an LDAP login or a SAML identifier for the group.
- `role_ids`: An array of role IDs defining the roles that you want to assign to users in this group. An empty array might be valid, but users can't do anything in PE if they are not assigned to any roles.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/groups" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"login": "augmentators",
```

```
"role_ids": [1,2,3}]'
```

Response format

If the create operation is successful, the endpoint returns 201 Created with a location header that points to the new resource.

Error responses

If the login for the group conflicts with an existing group login, the endpoint returns a 409 Conflict response.

For other errors, refer to [RBAC service errors](#) on page 313.

PUT /groups/<sid>

Replaces the content of the specified user group object. For example, you can update the group's roles or membership. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using all keys supplied in the [GET /groups/<sid>](#) on page 328 endpoint response.

You must supply all keys; however, the only key you can update is role_ids. Any values supplied in role_ids replace the group's current role values. If you want to add roles, you need to supply all of the group's *current* role IDs plus the new role IDs. If role_ids is empty, all roles are removed from the group (and, by extension, the roles are removed from users who belong to this group). Changes to keys other than role_ids are ignored.

Example curl request:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/groups/75370a30-588a-11e4-8ed6-0800200c9a66" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"id": "75370a30-588a-11e4-8ed6-0800200c9a66",
  "login": "admins",
  "display_name": "Admins",
  "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, the endpoint returns a 200 OK response and a group object with updated roles.

For errors, refer to [RBAC service errors](#) on page 313.

DELETE /groups/<sid>

Deletes the user group with the specified ID from PE RBAC. This endpoint does not change the directory service. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the URI path must include the ID of the group you want to delete from the PE console. For example:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v1/groups/75370a30-588a-11e4-8ed6-0800200c9a66" \
-H "X-Authentication:$(puppet-access show)"
```

Response format

If the user group is successfully deleted, the endpoint returns a 204 No Content response.

Error responses

If the requesting user does not have the `user_groups:delete` permission for this user group, the endpoint returns a 403 Forbidden response.

If there is no user group with the specified ID, the endpoint returns a 404 Not Found response.

For other errors, refer to [RBAC service errors](#) on page 313.

User roles endpoints

User roles contain sets of permissions. When you assign a user (or a user group) to a role, you can assign the entire set of permissions at once. This is more organized and easier to manage than assigning individual permissions to individual users. Use the `roles` endpoints to manage roles.

You can:

- [GET /roles](#) on page 331: Get a list of roles.
- [GET /roles/<rid>](#) on page 331: Get information about a specific role.
- [POST /roles](#) on page 332: Create a role.
- [PUT /roles/<rid>](#) on page 333: Edit a role.
- [DELETE /roles/<rid>](#) on page 334: Delete a role.
- [POST /command/roles/add-users](#) on page 334: Assign a role to users.
- [POST /command/roles/remove-users](#) on page 335: Remove a role from users.
- [POST /command/roles/add-groups](#) on page 335: Assign a role to user groups.
- [POST /command/roles/remove-groups](#) on page 336: Remove a role from user groups.
- [POST /command/roles/add-permissions](#) on page 336: Add permissions to a role.
- [POST /command/roles/remove-permissions](#) on page 337: Remove permissions from a role.

Tip: You'll want to be familiar with the [User roles endpoints keys](#) on page 330 that appear in these endpoints' requests and responses.

Some command endpoints are similar to other endpoints, such as the [POST /command/users/add-roles](#) on page 324 endpoint. However, in this case, the role is the focus. For example, whereas the [POST /command/users/add-roles](#) on page 324 endpoint assigns multiple roles to one user, the [POST /command/roles/add-users](#) on page 334 endpoint assigns one role to multiple users.

User roles endpoints keys

These keys are used with the RBAC API v1 `roles` endpoints.

Key	Definition	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"
<code>permissions</code>	An array containing permission objects that indicate what permissions a role grants. An empty array is valid. See Permissions endpoints keys on page 338 for possible content.	[]

Key	Definition	Example
user_ids	An array of UUIDs indicating which users and groups are directly assigned to the role. An empty array is valid. Users belonging to specified groups receive the role through the group, but those users aren't listed individually.	["fc115750-555a-11e4-916c-0800200c9a66"]
group_ids		

GET /roles

Fetches information about all user roles. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/roles" -H "X-Authentication:$(puppet-access show)"
```

Response format

The response is a JSON object that lists metadata for roles, including permissions objects, and lists of users and groups assigned to the role. 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" },
  ... ]
```

For information about keys in the response, refer to [User roles endpoints keys](#) on page 330.

For information about error responses, refer to [RBAC service errors](#) on page 313.

GET /roles/<rid>

Fetches information about a single role identified by its role ID (`rid`). Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, provide authentication and specify a role ID, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/roles/1234" -H "X-Authentication:$(puppet-access show)"
```

Response format

Returns a 200 OK response with a JSON object containing information about the requested role. 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" }

```

For information about keys in the response, refer to [User roles endpoints keys](#) on page 330.

For error responses, refer to [RBAC service errors](#) on page 313.

POST /roles

Create a role. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `permissions`: An array of permission objects (consisting of sets of `object_type`, `action`, and `instance`) defining the permissions associated with this role. Required, but can be empty. An empty array means no permissions are associated with the role.
- `group_ids`: An array of group IDs defining the user groups you want to assign this role to. All users in the groups (or added to the groups in the future) receive this role through their group membership. Required, but can be empty. An empty array means the role is not assigned to any groups.
- `user_ids`: An array of user IDs defining the individual users that you want to assign this role to. You do not need to repeat any users who are part of a group mentioned in `group_ids`. Required, but can be empty. An empty array means the role is not assigned to any individual users.
- `display_name`: A string naming the role.
- `description`: A string describing the role's purpose. Can be null.

For example:

```

curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/roles"
 \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
  "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" }'

```

For more information about these keys, refer to [User roles endpoints keys](#) on page 330.

Tip:

If you're writing a role for a task-target, you must include unique `action` and `instance` key values to specify permissions:

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.

For the complete task-target workflow, refer to the [Puppet Enterprise RBAC API](#), or how to manage access to tasks blog post.

Response format

If the role was successfully created, the endpoint returns a 201 Created response with a location header pointing to the new resource.

Tip: If your request included any empty arrays, you can use these endpoints to add permissions, groups, and users to your role:

- [POST /command/roles/add-permissions](#) on page 336
- [POST /command/roles/add-groups](#) on page 335
- [POST /command/roles/add-users](#) on page 334

Error responses

Returns a 409 Conflict response if the role has a name that collides with an existing role.

For other errors, refer to [RBAC service errors](#) on page 313.

Related information

[POST /command/task_target](#) on page 691

Create a *task-target*, which is a set of tasks and nodes/node groups you can use to provide specific privilege escalation for users who would otherwise not be able to run certain tasks or run tasks on certain nodes or node groups. When you grant a user permission to use a task-target, the user can run the task(s) in the task-target on the set of nodes defined in the task-target.

PUT /roles/<rid>

Replaces the content of the specified role object. For example, you can update the role's permissions or user membership. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using all keys supplied in the [GET /roles/<rid>](#) on page 331 endpoint response. You must supply all keys; however, you can't update the id key. Any values supplied in editable keys replace the role's current values. If you want to values, you need to supply all of the role's *current* values plus the new values. If you supply an empty array (or a null description), the current content is removed. For example, supplying an empty user_ids array removes any individual users that were assigned to the role.

For example:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/<rid>" \
-H "X-Authentication: $(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
    "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

If the operation is successful, the endpoint returns a 200 OK response with the updated role object.

Error responses

For errors, refer to [RBAC service errors](#) on page 313.

DELETE /roles/<rid>

Deletes the role with the specified role ID. Users who had this role (either directly or through a user group) immediately lose the role and all permissions granted by it, but their session is otherwise unaffected. The next action the user takes in PE is determined by their permissions without the deleted role.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the URI path must include the ID of the role you want to delete. For example:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v1/
roles/1234" \
-H "X-Authentication:$(puppet-access show)"
```

Response format

Returns a 200 OK response if the role was deleted.

Error responses

Returns a 404 Not Found response if no role exists for the specified role ID.

Returns a 403 Forbidden response if the requesting user lacks permission to delete the role identified by role ID.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/add-users

Assign a role to one or more users.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to assign users to.
- `user_ids`: An array of user IDs defining the users that you want to assign to the role.

Example payload:

```
{ "role_id": <role_id>, "user_ids": [ <user_id1>, <user_id2>,
<user_id3>, ... ] }
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/roles/add-users" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"role_id": 1, "user_ids": ["5c1ab4b0-588b-11e4-8ed6-0800200c9a66"]}'
```

Tip: To assign multiple roles to a single user at once, use the [POST /command/users/add-roles](#) on page 324 endpoint.

Response format

Returns 204 No Content if the users are successfully assigned the role.

Returns 404 Not Found if any of the users don't exist.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/remove-users

Remove a role from one or more users.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to remove users from.
- `user_ids`: An array of user IDs defining the users that you want to remove from the role.

Example payload:

```
{ "role_id": <role_id>, "user_ids": [<user_id1>, <user_id2>, <user_id3>, ...]}
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/ \
command/roles/remove-users" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"role_id": 1, "user_ids": ["5c1ab4b0-588b-11e4-8ed6-0800200c9a66"]}'
```

Response format

Returns 204 No Content if the users are removed from the role.

Note: A request with an invalid `role_id` still returns 204 No Content even though no users were removed from the nonexistent role.

Returns 400 Bad request if a user doesn't exist.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/add-groups

Add a role to one or more user groups.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to assign to groups.
- `group_ids`: An array of user group IDs defining the groups that you want to assign the role to.

Example payload:

```
{ "role_id": <role-id>, "group_ids": [<id>, <id>, <id>] }
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/ \
command/roles/add-groups" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"role_id": 1, "group_ids": ["2ca57e30-5887-11e4-8ed6-0800200c9a66"]}'
```

Response format

Returns 204 No Content if the user groups are successfully added to the role.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/remove-groups

Remove a role from one or more user groups.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to remove groups from.
- `group_ids`: An array of user group IDs defining the groups that you want to remove the role from.

Example payload:

```
{ "role_id": <role-id>, "group_ids": [1,2,3] }
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/roles/remove-groups" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"role_id": 1, "group_ids": ["a2450020-4217-439d-9bc4-258f6d2d7e76"]}'
```

Response format

Returns 204 No Content if the user groups are successfully removed from the role.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/add-permissions

Add permissions to a role.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to add permissions to.
- `permissions`: An array of permissions objects describing the permissions to add to the role. Permissions objects consist of sets of `object_type`, `action`, and `instance`.

Example payload:

```
{
  "role_id": <role-id>,
  "permissions": [
    { "object_type": <TYPE>,
      "action": <ACTION>,
      "instance": <INSTANCE> },
    ...
  ]
}
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/roles/add-permissions" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
```

```
-d '{"role_id": 1,
    "permissions": [
        {"object_type": "node_groups",
         "action": "edit_rules",
         "instance": "*"}
    ]
}'
```

Response format

Returns 204 No Content if the permissions are successfully added to the role.

For errors, refer to [RBAC service errors](#) on page 313.

POST /command/roles/remove-permissions

Remove permissions from a role.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `role_id`: The ID of the role you want to remove permissions from.
- `permissions`: An array of permissions objects describing the permissions to remove from the role. Permissions objects consist of sets of `object_type`, `action`, and `instance`.

Example payload:

```
{
  "role_id": <role-id>,
  "permissions": [
      {"object_type": <TYPE>,
       "action": <ACTION>,
       "instance": <INSTANCE>} ,
    ...
  ]}
```

Example curl request:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/roles/remove-permissions" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"role_id": 1,
    "permissions": [
        {"object_type": "node_groups",
         "action": "edit_rules",
         "instance": "*"}
    ]
}'
```

Response format

Returns 204 No Content when the permissions are successfully removed from the role.

An error in the standard format is returned for all other responses.

Permissions endpoints

You add permissions to roles to control what users can access and do in PE. Use the permissions endpoints to get information about objects you can create permissions for, what types of permissions you can create, and whether specific users can perform certain actions.

Tip: Before using these endpoints, you'll want to be familiar with [User permissions and user roles](#) on page 269, particularly [Best practices for assigning permissions](#) on page 277.

A permission consists of three components:

- Type (`object_type`)
- Permission (`action`)
- Object (`instance`, not to be confused with a JSON object)

These three components are described in [Structure of user permissions](#) on page 269, as well as the crucial `All` object ("*").

RBAC API requests and responses use the system names (not the display names) described in [Reference: User permissions and names](#) on page 270. This reference also provides helpful information about some permissions, such as some permissions that require the `All` ("*") object.

Permissions endpoints keys

These keys are used with the RBAC API v1 permissions endpoints.

Key	Definition	Example
<code>object_type</code>	A string identifying what PE object type the permission applies to, such as node groups, users, roles, and so on.	"node_groups"
<code>action</code>	A string indicating the permitted action, such as viewing, editing, or creating.	"modify_children"
<code>actions</code>	An array representing multiple actions, formatted as JSON objects.	Each JSON object contains: <ul style="list-style-type: none"> • <code>name</code>: The action's system name. • <code>display_name</code>: The action's name as it appears in the PE console. • <code>description</code>: • <code>has_instances</code>: Boolean indicating whether you can apply <code>instance</code> specification to this action. If <code>false</code>, you must supply "*" for the <code>instance</code> when including the action in a permission JSON object. Refer to <code>instance</code> for more information.

Key	Definition	Example
instance	<p>A string describing the scope of the permission.</p> <p>To apply the permission to all instances of the specified <code>object_type</code>, use "*" to indicate all instances.</p> <p>To limit the permission to specific instances of the specified <code>object_type</code>, supply the appropriate UUID, such as a specific node group ID or user ID.</p> <p>For any <code>object_type</code> that doesn't allow instance specification, you must supply "*".</p>	<ul style="list-style-type: none"> • To permit all instances (or if the <code>object_type</code> doesn't support instance specification): "*" • To define a specific instance, supply a UUID as a string, such as: "cec7e830-555b-11e4-916c-0800200c9a00"
display_name	A string containing the <code>object_type</code> name as it appears in the PE console.	"Node Groups"
description	A string describing an <code>object_type</code> .	"Groups that nodes can be assigned to."
token	In the POST /permitted on page 340 endpoint, this is a string representing the UUID of a user or user group.	"cec7e830-555b-11e4-916c-0800200c9a00"

Tip: You'll use `object_type`, `action`, and `instance` to build permissions. Use the [GET /types](#) on page 339 endpoint to get values you can use for these keys when writing permissions. For `object_type` and `action`, you must use system names, not display names.

Related information

[Structure of user permissions](#) on page 269

User permissions are structured as a triple of *type*, *permission*, and *object*.

[Reference: User permissions and names](#) on page 270

This reference describes the permissions granted to the five default Puppet Enterprise (PE) user roles, as well as the *display name* and *system name* for each type and permission.

GET /types

Lists each `object_type` that you can regulate with RBAC permissions, the available actions for each type, and whether each action allows instance specification. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/types" -H "X-Authentication:$(puppet-access show)"
```

Response format

The response is 200 OK and an array of object_type objects and actions for each type. 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
  }, ... ]
}, ... ]
```

For information about response keys and instance specification, refer to [Permissions endpoints keys](#) on page 338.

Error responses

Returns a 401 Unauthorized response if authentication is invalid.

Returns a 403 Forbidden response if the requesting user lacks permission to view types.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /permitted

Query whether a user or user group can perform specified actions. Use this to check if a user or group already has a certain permission. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object specifying a user or user group and one or more specific permissions. You must use these keys:

- token: The UUID of a user or user group.
- permissions: An array of JSON objects representing permissions. Each permissions object includes the object_type, action, and instance keys. For more information about these keys and how to populate them, refer to [Permissions endpoints keys](#) on page 338.

Tip: For any object_type that doesn't support instance specification, you must supply "instance": "*" .

For example, the following body queries a user or group with the UUID 456. It checks if this user or group can edit rules for a specific node group, and if the user or group can disable all ("*") user accounts. The response returns an array of Boolean values representing each JSON object in the permissions array. A true response indicates that the user or group can perform the specified action, whereas false indicates the user or group can't perform that action.

```
{ "token": "456",
  "permissions": [ { "object_type": "node_groups",
    "action": "edit_rules",
    "instance": "<NODE_GROUP_UUID" },
    { "object_type": "users",
      "action": "disable",
      "instance": "*" } ]
}
```

Here is an example of a complete curl request to the `permitted` endpoint:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/permitted" \
-H "X-Authentication: 0RrYy80wuJZLNDc0wxn119DJ Ae7LAkpPZtCbx8Jh3NxQ" \
-H "Content-type: application/json" \
-d '{"token": "42bf351c-f9ec-40af-84ad-e976fec7f4bd", \
"permissions": [{"object_type": "node_groups", \
"action": "edit_rules", \
"instance": "4"}]}'
```

Response format

If the request is well-formed and valid, the endpoint returns a 200 OK response with an array of Boolean values representing each JSON object in the `permissions` array in the request body.

The response array has the same length as the request's `permissions` array. Each returned Boolean value corresponds to the submitted permission query at the same index. For example, if you query two permissions, the response array contains two values, such as:

```
[true, false]
```

A `true` response indicates that the user or group can perform the specified action (described at the corresponding index position in the request), whereas `false` indicates the user or group can't perform that action.

The response is based on a full evaluation of permissions, including inherited roles and matching general permissions to more specific queries. For example, a query for `users:edit:1` returns `true` if the `token` subject has either permission to edit that specific user (`users:edit:1`) or permission to edit all users (`users:edit:*`).

For error responses, refer to [RBAC service errors](#) on page 313.

GET /permitted/<object-type>/<action>

For a specific `object_type` and `action`, get a list of `instance` IDs that the current authenticated user is permitted to take the specified action on. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the URI path must include the name of an `object_type` and applicable `action` for that object type. For example, this request refers to the `node_groups` type and the `view` action:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/permitted/node_groups/view" \
-H "X-Authentication: $(puppet-access show)"
```

Tip: This endpoint checks permissions for the current authenticated user. If you want to check permissions for another user, use the [GET /permitted/<object-type>/<action>/<uuid>](#) on page 342 endpoint.

Response format

A valid request returns 200 OK and an array of `instance` IDs that the authenticated user is permitted to perform the supplied `action` on. For example, this response has one instance:

```
[ "00000000-0000-4000-8000-000000000000" ]
```

If the user does not have permission to act on any instance, an empty array is returned.

Error responses

Returns 404 Not Found if:

- The supplied `object_type` does not map to a known `object_type`. Make sure your request used the type's system name, not the display name. System names are listed in [Reference: User permissions and names](#) on page 270.
- The supplied `action` does not exist for the given `object_type`. You can use the [GET /types](#) on page 339 endpoint to get a list of actions for each `object_type`.

For other errors, refer to [RBAC service errors](#) on page 313.

GET /permitted/<object-type>/<action>/<uuid>

For a specific `object_type` and `action`, get a list of instance IDs that the specific user (identified by UUID) is permitted to take the specified `action` on. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the URI path must include the name of an `object_type`, an applicable `action` for that object type, and a user's UUID. For example, this request checks if a specific user can take the `view` action on node groups:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/permitted/
node_groups/view/42bf351c-e976fec7f4bd" \
-H "X-Authentication:$(puppet-access show)"
```

Response format

A valid request returns 200 OK and an array of instance IDs that the specified user is permitted to perform the supplied `action` on. For example, this response has one instance:

```
[ "00000000-0000-4000-8000-000000000000" ]
```

If the user does not have permission to act on any instance, an empty array is returned.

Error responses

Returns 404 Not Found if:

- The supplied `object_type` does not map to a known `object_type`. Make sure your request used the type's system name, not the display name. System names are listed in [Reference: User permissions and names](#) on page 270.
- The supplied `action` does not exist for the given `object_type`. You can use the [GET /types](#) on page 339 endpoint to get a list of actions for each `object_type`.
- The `uuid` does not map to a known user.

For other errors, refer to [RBAC service errors](#) on page 313.

Tokens endpoints

Authentication tokens control access to PE services. Use the `auth/token` and `tokens` endpoints to create tokens.

You can use the v2 [Tokens endpoints](#) on page 363 to revoke or validate tokens.

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

Tokens endpoints keys

These keys are used with the tokens endpoints.

Key	Description
login	The user's login for the PE console. Only valid with POST /auth/token on page 343.
password	The user's password for the PE console. Only valid with POST /auth/token on page 343.
lifetime	Used to Set a token-specific lifetime on page 308. If omitted, the default token lifetime is used.
description	Optional description of the token.
client	Optional description about the client making the token request, such as PE console.
label	Optional key used to Set a token-specific label on page 309.

POST /auth/token

Generate an authorization token for a user identified by login and password. This token can be used to authenticate requests to Puppet Enterprise (PE) services, such as by using an X-Authentication header or a token query parameter in an API request.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object identifying the user you want to create a token for and optional token settings. You must supply the login and password keys. Other keys, such as the lifetime or label, are optional. For descriptions of keys, refer to [Tokens endpoints keys](#) on page 343.

Note: This endpoint accepts only login and password credentials for authentication. Generating a token using SSO credentials is not supported.

For example:

```
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"
```

Tip: This route is intended to require zero authentication to generate the key. While HTTPS is still required (unless PE is explicitly configured to permit HTTP), neither an allowed cert nor a session cookie is required to post to this endpoint.

The --cacert <FILE> argument in the above curl command specifies an authentication certificate as described in [Forming RBAC API requests](#) on page 311. Alternatively, at your discretion and understanding of the risks, you could use the -k or --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, you might get an error or warning about not being able to verify the RBAC server.

Here is an additional curl request example:

```
curl --insecure -X POST "https://$(puppet config print server):4433/rbac-api/v1/auth/token" \
-H "Content-type: application/json" \
-d '{"login": "test",
"password": "Test123!",
"lifetime": "4m",
"label": "personal workstation token"}'
```

Response format

If the credentials are valid and the specified user is not revoked, the endpoint returns 200 OK and the new token, such as:

```
{ "token": "0QX-WR3kgP0R9C2dA0I2nfnp0QgAT95_xH3iy1BhqroA" }
```

From here, you can save the token, as described in [Generate a token using the RBAC API](#) on page 306.

Error responses

Returns 401 Unauthorized if the credentials are invalid or the user is revoked.

Returns 400 Malformed if something is wrong with the request body.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /tokens

Create a token for the authenticated user. Doesn't allow certificate authentication.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body can be a JSON object identifying token settings, such as the lifetime or label. For descriptions of keys, refer to [Tokens endpoints keys](#) on page 343.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/tokens" \
-X "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"lifetime": "1y",
"description": "A token to be used with joy and care.",
"client": "PE console"}'
```

Response format

If the supplied authentication is valid, not expired, and the attached user is not revoked, the endpoint returns 200 OK and the new token, such as:

```
{ "token": "0QX-WR3kgP0R9C2dA0I2nfnp0QgAT95_xH3iy1BhqroA" }
```

Error responses are similar to the [POST /auth/token](#) on page 343 error responses.

LDAP endpoints

Use the LDAP ds (directory service) endpoints to get information about your LDAP directory service, test your LDAP directory service connection, and replace LDAP directory service connection settings.

Tip: To connect to the directory service anonymously, set the **Lookup user** and **Lookup password** fields to null or leave them blank.

Related information

[External directory settings](#) on page 282

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 (deprecated)

Get the connected directory service information. Authentication is required.

Important: The GET /v1/ds endpoint is deprecated. Instead, use the [GET /v2/ds endpoint](#).

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/ds" -H "X-Authentication:$(puppet-access show)"
```

Response format

Returns 200 OK with a JSON object representing the connection settings. For example:

```
{
  "help_link": "https://help.example.com",
  "ssl": true,
  "group_name_attr": "name",
  "password": <password>,
  "group_rdn": null,
  "connect_timeout": 15,
  "user_display_name_attr": "cn",
  "disable_ldap_matching_rule_in_chain": false,
  "ssl_hostname_validation": true,
  "hostname": "ldap.example.com",
  "base_dn": "dc=example,dc=com",
  "user_lookup_attr": "uid",
  "port": 636,
  "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
  "group_lookup_attr": "cn",
  "group_member_attr": "uniqueMember",
  "ssl_wildcard_validation": false,
  "user_email_attr": "mail",
  "user_rdn": "ou=users",
  "group_object_class": "groupOfUniqueNames",
  "display_name": "Acme Corp Ldap server",
  "search_nested_groups": true,
  "start_tls": false
}
```

Returns 200 OK with an empty JSON object ({}) if the connection settings are not specified.

For information about each setting, refer to [External directory settings](#) on page 282.

For errors, refer to [RBAC service errors](#) on page 313.

GET /ds/test

Test the connection to the connected directory service. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/ds/test" -H "X-Authentication:$(puppet-access show)"
```

Response format

If the test is successful, the response is 200 OK and a JSON object containing the directory service connection settings. For example:

```
{
  "help_link": "https://help.example.com",
  "ssl": true,
  "group_name_attr": "name",
  "password": <password>,
  "group_rdn": null,
  "connect_timeout": 15,
  "user_display_name_attr": "cn",
  "disable_ldap_matching_rule_in_chain": false,
  "ssl_hostname_validation": true,
  "hostname": "ldap.example.com",
  "base_dn": "dc=example,dc=com",
  "user_lookup_attr": "uid",
  "port": 636,
  "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
  "group_lookup_attr": "cn",
  "group_member_attr": "uniqueMember",
  "ssl_wildcard_validation": false,
  "user_email_attr": "mail",
  "user_rdn": "ou=users",
  "group_object_class": "groupOfUniqueNames",
  "display_name": "Acme Corp Ldap server",
  "search_nested_groups": true,
  "start_tls": false
}
```

For information about each setting, refer to [External directory settings](#) on page 282.

For errors, refer to [RBAC service errors](#) on page 313.

Error responses

Returns 400 Bad Request if the request is malformed.

Returns 401 Unauthorized if no user is authenticated.

Returns 403 Forbidden if the current user lacks permission to test the directory settings.

The error response also includes the elapsed time, such as { "elapsed": 20, "error": "..." }.

For other errors, refer to [RBAC service errors](#) on page 313.

PUT /ds/test

Tests a directory service connection based on supplied settings, rather than stored settings. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using all directory service setting keys. For example:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/ds/test" \
-H "X-Authentication: 0F4DITVB7HP3z8YnD95kx1W1jY0z5Pnc3ixB5uGAXzLY" \
-H "Content-type: application/json" \
-d '{
  "help_link": "https://help.example.com",
  "ssl": true,
  "group_name_attr": "name",
  "password": <password>,
  "group_rdn": null,
  "connect_timeout": 15,
  "user_display_name_attr": "cn",
  "disable_ldap_matching_rule_in_chain": false,
  "ssl_hostname_validation": true,
  "hostname": "ldap.example.com",
  "base_dn": "dc=example,dc=com",
  "user_lookup_attr": "uid",
  "port": 636,
  "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
  "group_lookup_attr": "cn",
  "group_member_attr": "uniqueMember",
  "ssl_wildcard_validation": false,
  "user_email_attr": "mail",
  "user_rdn": "ou=users",
  "group_object_class": "groupOfUniqueNames",
  "display_name": "Acme Corp Ldap server",
  "search_nested_groups": true,
  "start_tls": false
}'
```

For information about each setting, refer to [External directory settings](#) on page 282.

Tip: If you have an LDAP connection configured, you can use the [GET /ds](#) on page 367 endpoint to retrieve the current settings object and use it as a template for your PUT /ds/test request.

Response format

If the test succeeds, the endpoint returns a JSON object with information about the test, such as the amount of time the test ran. For example: { "elapsed": 10 }

Error responses

If the test fails, the body contains the elapsed time and information about the failure: { "elapsed": 20, "error": "..." }.

For other errors, refer to [RBAC service errors](#) on page 313.

PUT /ds

Replace current directory service connection settings. You can update the settings or disconnect the service (by removing all settings). Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json.

To change the settings, the body must be a JSON object containing, at minimum, all required directory service setting keys.

Important: When changing any directory service settings, you must specify **all** required directory service settings in the request body, including required settings that you aren't changing. However, you don't need to specify optional settings unless you are changing them.

If you omit a required setting, the setting is removed or reset to the default value.

All [External directory settings](#) on page 282 are required except `help-link`, `login`, `password`, `user_rdn`, and `group_rdn`. However, your specific LDAP configuration might require some of these fields, in which case you must treat those fields as required fields.

Here is an example curl request to change settings:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/ds" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{ "help_link": "https://help.example.com",
      "ssl": true,
      "group_name_attr": "name",
      "password": <password>,
      "group_rdn": null,
      "connect_timeout": 15,
      "user_display_name_attr": "cn",
      "disable_ldap_matching_rule_in_chain": false,
      "ssl_hostname_validation": true,
      "hostname": "ldap.example.com",
      "base_dn": "dc=example,dc=com",
      "user_lookup_attr": "uid",
      "port": 636,
      "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
      "group_lookup_attr": "cn",
      "group_member_attr": "uniqueMember",
      "ssl_wildcard_validation": false,
      "user_email_attr": "mail",
      "user_rdn": "ou=users",
      "group_object_class": "groupOfUniqueNames",
      "display_name": "Acme Corp Ldap server",
      "search_nested_groups": true,
      "start_tls": false}'
```

If you want to disconnect the directory service from PE, you can supply an empty object ({}) or set all required settings set to null.

Tip: If you have an LDAP connection configured, you can use the [GET /ds](#) on page 367 endpoint to retrieve the current settings object and use it as a template for your `PUT /ds` request. This also helps avoid accidentally omitting a setting.

Searching nested groups

When authorizing users, the RBAC service can search nested groups. Nested groups are groups that belong to external directory groups. For example, assume your external directory has a System Administrators group, and you've given that group a Superusers user role in RBAC. In addition to assigning the Superusers role to individual users in the System Administrators group, RBAC looks for other groups in the System Administrators group and assigns the Superusers role to the individual users in those nested groups.

By default, RBAC does not search nested groups. To enable nested group searches, set `search_nested_groups` to `true`.

Important: This setting causes RBAC to search the entire group hierarchy when users log in; therefore, you might experience slowdowns in performance if you have a lot of nested groups. To avoid these performance issues, set `search_nested_groups` to `false`. This disables nested group searches so RBAC only searches the groups it is configured to use for user roles.

Note: In Puppet Enterprise (PE) versions 2015.3 and earlier, RBAC searched nested groups by default. If you upgrade from one of these earlier versions, this setting is preserved and RBAC continues to search nested groups by default. You'll need to disable it (by setting `search_nested_groups` to `false`) if you don't want to use nested searching anymore.

Using StartTLS connections

You can set `start_tls` to `true` to use StartTLS to secure the connection to the directory service. Any certificates you configured through the DS trust chain setting are used to verify the identity of the directory service. If you set `start_tls` to `true`, make sure `ssl` is set to `false`.

Disabling matching rule in chain

When PE detects an Active Directory that supports the `LDAP_MATCHING_RULE_IN_CHAIN` feature, PE automatically uses it. Under specific circumstances, you might need to disable this setting by setting `disable_ldap_matching_rule_in_chain` to `true`. Otherwise, this setting is optional.

Response format

Returns 200 OK with an object showing the updated connection settings. For example:

```
{
  "help_link": "https://help.example.com",
  "ssl": true,
  "group_name_attr": "name",
  "password": <password>,
  "group_rdn": null,
  "connect_timeout": 15,
  "user_display_name_attr": "cn",
  "disable_ldap_matching_rule_in_chain": false,
  "ssl_hostname_validation": true,
  "hostname": "ldap.example.com",
  "base_dn": "dc=example,dc=com",
  "user_lookup_attr": "uid",
  "port": 636,
  "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
  "group_lookup_attr": "cn",
  "group_member_attr": "uniqueMember",
  "ssl_wildcard_validation": false,
  "user_email_attr": "mail",
  "user_rdn": "ou=users",
  "group_object_class": "groupOfUniqueNames",
  "display_name": "Acme Corp Ldap server",
  "search_nested_groups": true,
  "start_tls": false
}
```

For errors, refer to [RBAC service errors](#) on page 313.

SAML endpoints

Use the `saml` endpoints to configure SAML, retrieve SAML configuration details, and get the public certificate and URLs needed for configuration.

PUT /saml

Use this endpoint to configure SAML. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object containing, at minimum, all required SAML setting keys.

Important: When changing any SAML settings, you must specify **all** required settings in the request body, including required settings that you aren't changing. However, you don't need to specify optional settings unless you are changing them.

If you omit a required setting, the setting is removed or reset to the default value.

The [SAML configuration reference](#) on page 291 indicates which settings are required or optional. However, your specific SAML configuration might require some of the optional settings, in which case you must treat those settings as required settings.

Example curl request:

```
curl -X PUT "https://$(puppet config print server):4433/rbac-api/v1/saml" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{"display_name": "Corporate Okta",
  "idp_sso_url": "https://idp.example.org/SAML2/SSO",
  "idp_slo_url": "https://ipd.example.com/SAML2/SLO",
  "idp_certificate": [<certificate>],
  "want_messages_signed": true,
  "want_assertions_signed": true,
  "sign_metadata": true,
  "want_assertions_encrypted": true,
  "want_name_id_encrypted": true,
  "allow_duplicated_attribute_name": true,
  "want_xml_validation": true,
  "signature_algorithm": "rsa-sha256",
  "requested_authn_context_comparison": "exact",
  "user_display_name_attr": "test",
  "user_lookup_attr": "test_lookup",
  "requested_auth_context": "test-request",
  "group_lookup_attr": "group_lookup_test",
  "user_email_attr": "email_attr",
  "idp_entity_id": "entity_id"}'
```

Tip: If you already have a SAML configuration, you can use the [GET /saml](#) on page 351 endpoint to retrieve the current settings object and use it as a template for your `PUT /saml` request. This also helps avoid accidentally omitting a setting.

Response format

If you provided new settings, the endpoint returns `201 Created` and the new settings. For example:

```
{ "want_xml_validation":true,
  "sign_metadata":true,
  "requested_authn_context_comparison": "exact",
  "want_assertions_encrypted":true,
  "want_name_id_encrypted":true,
```

```

"want_messages_signed":true,
"signature_algorithm":"rsa-sha256",
"user_display_name_attr":"test",
"want_assertions_signed":true,
"user_lookup_attr":"test_lookup",
"requested_auth_context":"test-request",
"allow_duplicated_attribute_name":true,
"idp_sso_url":"https://idp.example.org/SAML2/SSO",
"group_lookup_attr":"group_lookup_test",
"idp_certificate":["MIIGADCCA
+igAwIBAgIBAjANBgkqhkiG9w0BAQsFADBqMWgwZgYDVQQDDF9QdXBw"] ,
"user_email_attr":"email_attr",
"display_name":"Corporate Okta",
"idp_entity_id":"entity_id",
"idp_slo_url":"https://ipd.example.com/SAML2/SLO"
}

```

Returns 200 OK if you changed existing settings, and the changes were applied successfully.

Returns 400 Bad Request if the request was missing required settings. The [SAML configuration reference](#) on page 291 specifies required settings.

Returns 403 Forbidden if the user lacks the `directory_serivce:edit:*` permission.

For other errors, refer to [RBAC service errors](#) on page 313.

GET /saml

Retrieves the current SAML configuration settings. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/saml" -H "X-Authentication:$(puppet-access show)"
```

Response format

If the authentication is valid and there is an existing SAML configuration, the endpoint returns 200 OK and a JSON object containing the current SAML configuration settings. For example:

```
{
"want_xml_validation":true,
"sign_metadata":true,
"requested_authn_context_comparison":"exact",
"want_assertions_encrypted":true,
"want_name_id_encrypted":true,
"want_messages_signed":true,
"signature_algorithm":"rsa-sha256",
"user_display_name_attr":"test",
"want_assertions_signed":true,
"user_lookup_attr":"test_lookup",
"requested_auth_context":"test-request",
"allow_duplicated_attribute_name":true,
"idp_sso_url":"https://idp.example.org/SAML2/SSO",
"group_lookup_attr":"group_lookup_test",
"idp_certificate":["MIIGADCCA
+igAwIBAgIBAjANBgkqhkiG9w0BAQsFADBqMWgwZgYDVQQDDF9QdXBw"] ,
"user_email_attr":"email_attr",
"display_name":"Corporate Okta",
"idp_entity_id":"entity_id",
"idp_slo_url":"https://ipd.example.com/SAML2/SLO"
```

```
}
```

Returns 404 Not Found if the SAML data is not configured.

For information about each setting, refer to [SAML configuration reference](#) on page 291.

For errors, refer to [RBAC service errors](#) on page 313.

DELETE /saml

Remove the current SAML configuration. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v1/saml"
-H "X-Authentication:$(puppet-access show)"
```

Response format

Returns 204 No Content if the SAML configuration is removed successfully.

Returns 404 Not Found if no SAML configuration was set prior to making the DELETE request.

Returns 403 Forbidden if the user lacks the `directory_service:edit*` permission.

For other errors, refer to [RBAC service errors](#) on page 313.

GET /v1/saml/meta

Retrieve the public SAML certificate and URLs you need to configure an identity provider. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/saml/meta" -H
"X-Authentication:$(puppet-access show)"
```

Response format

If the instance is not a replica and the certificate exists, the endpoint returns 200 OK and an object containing these keys:

Key	Definition
meta	A URL to the public metadata endpoint for the SAML service provider. Some IdP configurations also require this URL in the <code>entity_id</code> and/or <code>audience_restriction</code> 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 ..."
}

```

Use these values to configure your identity provider. After configuration, your identity provider supplies the required values for configuring SAML in Puppet Enterprise (PE). You can also see this information in the PE console on the **SSO** tab.

Error response

Returns 404 Not Found if the public key file doesn't exist or the SAML key entries aren't present in the configuration.

For other errors, refer to [RBAC service errors](#) on page 313.

Passwords endpoints

When local users forget their Puppet Enterprise (PE) 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. Use the `password` endpoints to generate password reset tokens, use tokens to reset passwords, change the authenticated user's password, and validate potential user names and passwords.

Important: The password endpoints are for managing local user accounts within PE. You can't use these endpoints to modify user information in SAML or LDAP.

Tip: By default, users can make 10 login attempts before being locked out. You can change the amount of allowed attempts by configuring the `failed-attempts-lockout` parameter.

You can reset the PE console admin password with a password reset script available on the PE console node.

Related information

[Creating and managing local users and user roles](#) on page 278

Role-based access control (RBAC) in Puppet Enterprise (PE) lets you manage users—what they can and can't create, edit, or view—in an organized, high-level way that is 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 directly to users in PE, only to user roles. You then assign roles to users.

[Configure RBAC and token-based authentication settings](#) on page 224

You can configure RBAC and token-based authentication settings, such as setting the number of failed attempts a user has before they are locked out of the console or the amount of time tokens are valid.

[Reset the console administrator password](#) on page 266

If you're unable to log in to the console as `admin`, you can change the password from the command line of the node that is running console services.

POST /users/<uuid>/password/reset

Generate a single-use, limited-lifetime password reset token for a specific local user. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, provide authentication and specify a user ID, such as:

```

curl -X POST "https://${(puppet config print server)}:4433/rbac-api/v1/
users/297f1d72-d96e/password/reset" \
-H "X-Authentication:${(puppet-access show)}"

```

Response format

A successful request returns 200 OK and the new token. Use this token with [POST /auth/reset](#) on page 354 to reset the user's password.

Restriction: Password reset tokens can be used only once, and these tokens have a limited lifetime. The lifetime is based on the value of the `rbac_password_reset_expiration` parameter. The default is 24 hours. For more information, refer to [Configure RBAC and token-based authentication settings](#) on page 224.

Error responses

Returns 403 Forbidden if:

- The requesting user does not have permission to create a reset token for the specified user.
- The specified user is a remote user. You must manage remote user information within the relevant remote system, such as SAML or LDAP.

Returns 404 Not Found if there is no user with the given UUID.

For other error responses, refer to [RBAC service errors](#) on page 313.

POST /auth/reset

Use a password reset token to change a local user's password. Authentication is not required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `token`: A password reset token obtained from the [POST /users/<uuid>/password/reset](#) on page 353 endpoint.
- `password`: A new password to assign to the user attached to the password reset token.

Authentication is not required.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/auth/reset" \
-H "Content-type: application/json" \
-d '{"token": "0F1AtJ-84LMswcyzC8h9c2Hkreq114W6UeWKJJScYUUk", \
"password": "W3lcome!"}'
```

The body doesn't explicitly identify the user, because the user is identified through the password reset token.

Response format

Returns 200 OK if the password reset token is valid and the password was successfully changed. The user can now log in with the new password.

This endpoint only resets the password; it does not establish a valid log-in session for the user.

Error responses

Returns 403 Forbidden if the password reset token was already used or has expired.

Remember: Password reset tokens can be used only once, and these tokens have a limited lifetime. The lifetime is based on the value of the `rbac_password_reset_expiration` parameter. The default is 24 hours. For more information, refer to [Configure RBAC and token-based authentication settings](#) on page 224.

For other errors, refer to [RBAC service errors](#) on page 313.

PUT /users/current/password

Changes the current authenticated local user's password. You must provide the current password in the request. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using the following keys:

- `current_password`: The authenticated user's current password.
- `password`: A new password to assign to the authenticated user.

Authentication is required.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/users/current/password" \
-H "Content-type: application/json" \
-H "X-Authentication: $(puppet access show)" \
-d '{"current_password": "old_password",
     "password": "new_password"}'
```

The body doesn't explicitly identify the user, because the user is identified through authentication.

Response format

Returns 204 No Content if the password was successfully changed. You can now log in with the new password.

This endpoint only resets the password; it does not establish a valid log-in session.

Error response

Returns 403 Forbidden if the authenticated user is a remote user or if `current_password` doesn't match the user's stored password.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/validate-password

Check whether a password is valid. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is `application/json`. The body must be a JSON object using one or both of these keys:

- `password`: A password string to validate for compliance with password format and complexity requirements.
- `reset-token`: If your request uses certificate authentication, you can provide a password reset token to identify the user for password validation. You can get password reset tokens from the [POST /users/<uuid>/password/reset](#) on page 353 endpoint. Otherwise you can use token-based authentication (as an X-Authentication header) to identify the relevant user.

Use this body format for a token-authenticated request:

```
{ "password": <password>}
```

Use this body format is for a certificate-authenticated request:

```
{
  "password": <password>,
  "reset-token": <reset_token>
}
```

For example, this request uses token-based authentication to identify a user:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/validate-password" \
-H 'Content-Type: application/json' \
-H "X-Authentication: $(puppet access show)" \
-d '{ "password": "password" }'
```

Response format

If the password is valid, the endpoint returns 200 OK and { "valid": true }.

If the password isn't valid, the endpoint returns 200 OK and information about why the password is not valid. For example:

```
{
  "valid": false,
  "failures": [
    {
      "rule-identifier": "letters-required",
      "friendly-error": "Passwords must have at least 2 letters."
    }
  ]
}
```

If the request has formatting issues, the endpoint returns 400 Bad Request.

For other errors, refer to [RBAC service errors](#) on page 313.

POST /command/validate-login

Check whether a user name (login) is valid. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object containing the login key, which specifies the user name to validate. For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
command/validate-login" \
-H 'Content-Type: application/json' \
-H "X-Authentication: $(puppet-access show)" \
-d '{ "login": "1" }'
```

Response format

If the user name is valid, the endpoint returns 200 OK and { "valid": true }.

If the user name is invalid, the endpoint returns 200 OK and information about why the username is not valid. For example:

```
{
  "valid": false,
  "failures": [
    {
      "rule-identifier": "login-minimum-length",
      "friendly-error": "The login for the user must be a minimum of 3
characters."
    }
  ]
}
```

If the request has formatting issues, the endpoint returns 400 Bad Request.

For other errors, refer to [RBAC service errors](#) on page 313.

Disclaimer endpoints

Use these endpoints to modify the disclaimer text that appears on the Puppet Enterprise (PE) console login page.

You can use a `disclaimer.txt` file to [Create a custom login disclaimer](#) on page 267; however, the disclaimer endpoints allow you to configure your custom login disclaimer message without needing to reference a specific file location on disk.

Important: These endpoints do not modify or interact with `disclaimer.txt` files.

If you provide disclaimer text through both a `disclaimer.txt` file and `POST /command/config/set-disclaimer`, PE uses the `set-disclaimer` text.

GET /config/disclaimer

Retrieve the current disclaimer text, as specified by `POST /command/config/set-disclaimer`. This endpoint does not retrieve the contents of any `disclaimer.txt` file.

Request format

You must have the `configuration:view_disclaimer` permission to use this endpoint.

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v1/config/disclaimer" -H "X-Authentication: $(puppet access show)"
```

Response format

If `POST /command/config/set-disclaimer` on page 358 was previously used to specify disclaimer text, a well-formed request returns 200 OK and the disclaimer text, such as:

```
{
  "disclaimer": "Not to be accessed by unauthorized users"
}
```

The endpoint return 404 Not Found if:

- You haven't specified disclaimer text with `POST /command/config/set-disclaimer` on page 358.
- Previously-specified text was removed with `POST /command/config/remove-disclaimer` on page 358.
- You've only used a `disclaimer.txt` file to [Create a custom login disclaimer](#) on page 267. The `GET /config/disclaimer` endpoint doesn't check for the existence of a `disclaimer.txt` file, and it doesn't return the contents of such a file.

If you lack permission to retrieve the disclaimer text, the response is 403 Not Permitted.

For other error responses, refer to [RBAC service errors](#) on page 313.

Related information

[User permissions and user roles](#) on page 269

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 with that role can or can't do within Puppet Enterprise (PE).

POST /command/config/set-disclaimer

Change the disclaimer text that is on the PE console login page.

Request format

This endpoint requires authentication, and the requesting user must have the configuration:edit_disclaimer permission.

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object containing the disclaimer key, which accepts string-formatted disclaimer text. For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
  command/config/set-disclaimer" \
-H 'Content-Type: application/json' \
-H "X-Authentication: $(puppet access show)" \
-d '{ "disclaimer": "Unauthorized access prohibited." }' \
```

Tip: To remove the disclaimer text, use [POST /command/config/remove-disclaimer](#) on page 358. Setting disclaimer to an empty string or whitespace-only string causes the **Disclaimer** banner to be present, but empty, on the console login page.

Response format

A successful request returns 204 No Content.

Error responses

Returns 403 Not Permitted if you don't have the configuration:edit_disclaimer permission.

Returns 400 Bad Request if the disclaimer value is not a string.

For other errors, refer to [RBAC service errors](#) on page 313.

Related information

[User permissions and user roles](#) on page 269

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 with that role can or can't do within Puppet Enterprise (PE).

POST /command/config/remove-disclaimer

Remove the disclaimer text set through POST /command/config/set-disclaimer.

Request format

This endpoint requires authentication, and the requesting user must have the configuration:edit_disclaimer permission.

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v1/
  command/config/remove-disclaimer" \
-H "X-Authentication: $(puppet access show)" \
```

Response format

A successful request returns 204 No Content and the **Disclaimer** banner is removed from the PE console login page.

However, if you had previously used a `disclaimer.txt` file to [Create a custom login disclaimer](#) on page 267, and the `disclaimer.txt` file still exists in the appropriate location, then PE falls back to this file and displays the content of this file on the console login page.

Error responses

Requests must contain the `Content-Type: application/json` header.

Returns 403 Not Permitted if you don't have the `configuration:edit_disclaimer` permission.

For other errors, refer to [RBAC service errors](#) on page 313.

Related information

[User permissions and user roles](#) on page 269

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 with that role can or can't do within Puppet Enterprise (PE).

RBAC API v2

The role-based access control (RBAC) API v2 service enables you to fetch information about users, create groups, revoke tokens, validate tokens, and get information about your LDAP directory service.

The v2 endpoints either extend or replace some [RBAC API v1](#) on page 316 endpoints.

- [Users endpoints](#) on page 359

With role-based access control (RBAC), you can manage local users and remote users (created on a directory service). Use the RBAC API v2 `GET /users` endpoint to get lists of users and information about users.

- [User group endpoints](#) on page 362

User groups allow you to quickly assign one or more roles to a set of users by placing all relevant users in the group. This is more efficient than assigning roles to each user individually. The v2 `POST /groups` endpoint has additional optional parameters you can use when creating groups.

- [Tokens endpoints](#) on page 363

Authentication tokens control access to PE services. Use the v2 `tokens` endpoints to revoke and validate tokens.

- [LDAP endpoints](#) on page 367

Use the v2 `ds` (directory service) endpoint to get information about the LDAP directory service connection.

Related information

[API index](#) on page 33

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

Users endpoints

With role-based access control (RBAC), you can manage local users and remote users (created on a directory service). Use the RBAC API v2 `GET /users` endpoint to get lists of users and information about users.

Use the RBAC API v1 [Users endpoints](#) on page 316 for other user functions, including creating, editing, deleting, revoking, and reinstating users.

Users endpoints keys

These keys appear in RBAC API v2 `GET /users` endpoint responses:

Key	Definition	Example
<code>id</code>	A UUID string identifying the user.	"4fee7450-54c7-11e4-916c-0800200c9a

Key	Definition	Example
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 roles to directly assign to the user. An empty array is valid.	[3 6 5]
is_group	These flags indicate whether a user is remote and/or a super user. For all users, <code>is_group</code> is always <code>false</code> .	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
last_login	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-0800200c9a8f"]

GET /users

Fetches all users, both local and remote (including the superuser) with options for filtering and sorting response content. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v2/users" -H "X-Authentication:$(puppet-access show)"
```

The default request fetches all users (limited to 500 records, sorted by subject ID in ascending order). You can append these optional query parameters to modify the response:

- `offset`: Specify a zero-indexed integer to retrieve user records starting from the offset point. The default is 0. This parameter is useful for omitting initial, irrelevant results, such as test data.
- `limit`: Specify a positive integer to limit the number of user records returned. The default is 500 records.
- `order`: Specify, as a string, whether records are returned in ascending (`asc`) or descending (`desc`) order. The default is `asc`. The `order_by` parameter specifies the basis for sorting.

- `order_by`: Specify, as a string, what information to use to sort the records. Choose from `login`, `email`, `display_name`, `last_login`, `id`, or `creation_date`. The default is `id`.
- `filter`: Specify a case-insensitive partial string. This parameter queries the `email`, `display_name`, and `login` fields. For example, `filter="example.com"` searches for users with `example.com` in any of those three fields.
- `include_roles`: Specify whether you want the response to include role information. The default is `false`.

To include parameters in your request, start with:

```
curl "https://$(puppet config print server):4433/rbac-api/v2/users?"
```

Then, add each parameter separated by an ampersand, such as:

```
limit=400&order="desc"
```

Enclose string values in double quotes. To form a complete request, make sure to close the single quote after your last parameter and include authentication details. For example:

```
curl "https://$(puppet config print server):4433/rbac-api/v2/users?
limit=400&offset=1&order="desc"&order_by="last_login"&filter="example.com"&include_roles=true"
 \
-H "X-Authentication:$(puppet-access show)"
```

Response format

The response is a JSON array of user objects followed by a copy of the request parameters. User objects contain role information only if you put `include_roles=true` in the request. For example, this response includes three records with role information:

```
{
  "users": [
    {
      "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"
}
],
"pagination": {
    "total": 1,
    "limit": 400,
    "offset": 1,
    "order": "desc",
    "filter": "example.com",
    "order_by": "last_login"
}
}

```

For information about user object keys, refer to [Users endpoints keys](#) on page 359. The pagination keys are described in the **Request format** section, above.

For information about error responses, refer to [RBAC service errors](#) on page 313.

User group endpoints

User groups allow you to quickly assign one or more roles to a set of users by placing all relevant users in the group. This is more efficient than assigning roles to each user individually. The v2 POST /groups endpoint has additional optional parameters you can use when creating groups.

Use the v1 [User groups endpoints](#) on page 326 to perform other user group functions, such as getting information about groups and deleting groups.

Remember: Group membership is determined by your directory service hierarchy. Therefore, local users (that exist only in the PE console) can't be in directory groups. You'll need to use the [Users endpoints](#) on page 316 to manage these users' roles.

POST /groups

Create a new remote directory user group. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using the following keys:

- `login`: The name to assign to the group.
- `role_ids`: An array of role IDs defining the roles that you want to assign to users in this group. An empty array might be valid, but users can't do anything in PE if they are not assigned to any roles.

The endpoint accepts a JSON body containing these keys:

Key	Definition
<code>login</code>	Required. Defines the group for an external IdP. This could be an LDAP login or a SAML identifier for the group.

Key	Definition
role_ids	Required. An array of role IDs defining the roles that you want to assign to users in this group. An empty array might be valid, but users can't do anything in PE if they are not assigned to any roles.
display_name	Optional. Specify a name for the group as you want it to appear in the PE console. If the group you're creating originates from an LDAP group, the LDAP group's Display name setting overrides this parameter.
validate	Optional. A Boolean specifying whether you want to validate if the group exists on the LDAP server prior to creating it. The default is <code>true</code> . Set this to <code>false</code> if you don't want to validate the group's existence in LDAP.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v2/groups"
  \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
    "login": "augmentators",
    "role_ids": [1,2,3],
    "display_name: "The Augmentators"
}'
```

Response format

If the new remote group is created successfully, the endpoint returns 303 See Other with a location header pointing to the new resource.

Error response

Returns 409 Conflict if the new group conflicts with an existing group.

For other errors, refer to [RBAC service errors](#) on page 313.

Tokens endpoints

Authentication tokens control access to PE services. Use the v2 tokens endpoints to revoke and validate tokens.

You can use the v1 [Tokens endpoints](#) on page 342 to create tokens.

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they 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 to access PE services.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, your request must use at least one of these query parameters to specify the tokens you want to revoke:

- `revoke_tokens`: Supply a list of complete authentication tokens you want to revoke. Any user can revoke any token by supplying the complete token in this parameter.
- `revoke_tokens_by_usernames`: Supply a list of user names identifying users whose tokens you want to revoke. To revoke tokens by user name, the user making the request must have the **Users Revoke** permission for the specified users.
- `revoke_tokens_by_labels`: Supply a list of labels identifying tokens to revoke. To be revoked in this manner, the tokens must belong to the requesting user and have been assigned a [token-specific label](#).
- `revoke_tokens_by_ids`: Supply a list of UUIDs for users whose tokens you want to revoke. To revoke tokens by user name, the user making the request must have the **Users Revoke** permission for the specified users.

You can append the parameters to the URI path, supply a JSON-encoded body, or both.

For example, this request uses an appended query parameter.

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v2/tokens?revoke_tokens_by_usernames=<USER_NAME>, <USER_NAME>" \
-H "X-Authentication:$(puppet-access show)"
```

When supplying parameters in a JSON-encoded body, specify the content type as `application/json`, format the entire body as a JSON object, and format each value list as an array. For example:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v2/tokens" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
  "revoke_tokens": [ "<TOKEN>", "TOKEN" ],
  "revoke_tokens_by_labels": [ "Workstation Token", "VPS Token" ],
  "revoke_tokens_by_usernames": [ "<USER_NAME>", "<USER_NAME>" ]
}'
```

When supplying multiple values, use commas to separate tokens, labels, user names, and IDs.

If you supply values by appending parameters and in a JSON body, the values from the both sources are combined.

It is not an error to specify the same token using multiple means. For example, you could supply the entire token to the `revoke_tokens` parameter and also include its label in the value of `revoke_tokens_by_labels`.

All operations on this endpoint are idempotent. It is not an error to revoke the same token multiple times.

Response format

The server sends a `204 No Content` response if all operations succeed.

Error responses

In the case of an error, malformed 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 while some requested operations succeed. For example, you would get multiple errors if a request included some malformed user names and a database error occurred when trying to revoke the well-formed user names.

Error codes include:

- `500 Application Error`: There was a database error when trying to revoke tokens.

- 403 Forbidden: The user lacks permission to revoke one of the supplied user names and no database error occurred.
- 400 Malformed: One of these conditions is true:
 - At least one of the tokens, user names, labels, or IDs is malformed.
 - At least one of the user names or IDs does not exist in the RBAC database.
 - The request contains no parameters or values to revoke.
 - The request contains illegal parameters.

In the error response, the `msg` key contains information about encountered errors and either `No tokens were revoked` or `All other tokens were successfully revoked`, depending on whether any operations were successful. For example:

```
"msg": "The following user does not exist: FormerEmployee. All other tokens were successfully revoked."
```

The error response also returns unprocessed or erroneous values in the `details` key in the error response. For example, this response had one failed operation and succeeded in all other operations:

```
{"kind": "malformed-request",
  "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}}
```

Error categories include:

- `malformed_tokens`, `malformed_usernames`, `malformed_labels`, and `malformed_ids`: Contain any values from the request that aren't properly formatted as tokens, user names, labels, or UUIDs.
- `nonexistent_usernames`: Contains any value from the `revoke_tokens_by_usernames` parameter that doesn't match an existing user's user name.
- `permission_denied_usernames` and `permission_denied_ids`: The requesting user doesn't have permission to revoke tokens for users identified in these arrays.
- `unrecognized_parameters`: The request contained an invalid or malformed parameter.

The `other_tokens_revoked` Boolean indicates whether any non-erroneous values were successfully revoked.

For more information about RBAC API errors and errors not described here, refer to [RBAC service errors](#) on page 313.

DELETE /tokens/<token>

Use this endpoint to revoke a single token, ensuring that it can no longer be used with RBAC. Authentication is required.

Request format

Only admins or API users can use this endpoint.

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication and a single token specified in the URI path. For example:

```
curl -X DELETE "https://$(puppet config print server):4433/rbac-api/v2/tokens/<TOKEN>" \
```

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

Tip: This endpoint is equivalent to using the [DELETE /tokens](#) on page 364 endpoint with the `revoke_tokens` parameter and a single token value. If you're not an admin, try the [DELETE /tokens](#) route for revoking tokens.

Response format

The server returns 204 No Content if the revocation was successful.

Error responses

Error response are similar to [DELETE /tokens](#) on page 364 error responses, except that only one token is processed.

POST /auth/token/authenticate

Use this endpoint to exchange a token for a map representing an RBAC subject and associated token data. Authentication isn't required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the content type is application/json. The body must be a JSON object using these keys:

- `token`: An authentication token
- `update_last_activity?`: A Boolean indicating whether you want a successful request to update the token's `last_active` timestamp.

For example:

```
curl -X POST "https://$(puppet config print server):4433/rbac-api/v2/auth/
token/authenticate" \
-H "X-Authentication:$(puppet-access show)" \
-H "Content-type: application/json" \
-d '{
    "token": "<TOKEN>" ,
    "update_last_activity?": false
}'
```

Response format

A successful request returns a 200 OK response and JSON object representing the RBAC subject and associated token data, such as:

```
{
  "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
}
```

For information about keys describing the user, refer to [Users endpoints keys](#) on page 317. For information about keys describing the token, refer to [Tokens endpoints keys](#) on page 343.

Error responses

Invalid requests return these errors:

- 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.

For other errors, refer to [RBAC service errors](#) on page 313.

LDAP endpoints

Use the v2 `ds` (directory service) endpoint to get information about the LDAP directory service connection.

Use the v1 [LDAP endpoints](#) on page 345 to test the connection and replace LDAP settings.

GET /ds

Get information about your directory service. Authentication is required.

Request format

When [Forming RBAC API requests](#) on page 311 to this endpoint, the request is a basic call with authentication, such as:

```
curl "https://$(puppet config print server):4433/rbac-api/v2/ds" -H "X-Authentication:$(puppet-access show)"
```

Response format

Returns an array of objects, where each object represents a currently-configured LDAP servers. For example, this response contains information for one LDAP server:

```
[
  {
    "id": "6e33eb78-820f-463a-a65c-e1ef291d59a8",
    "help_link": "https://help.example.com",
    "ssl": true,
    "group_name_attr": "name",
    "group_rdn": null,
    "connect_timeout": 15,
    "user_display_name_attr": "cn",
    "disable_ldap_matching_rule_in_chain": false,
    "ssl_hostname_validation": true,
    "hostname": "ldap.example.com",
    "base_dn": "dc=example,dc=com",
    "user_lookup_attr": "uid",
    "port": 636,
    "login": "cn=ldapuser,ou=service,ou=users,dc=example,dc=com",
    "group_lookup_attr": "cn",
    "group_member_attr": "uniqueMember",
    "ssl_wildcard_validation": false,
    "user_email_attr": "mail",
    "user_rdn": "ou=users",
    "group_object_class": "groupOfUniqueNames",
```

```

        "display_name": "Acme Corp Ldap server",
        "search_nested_groups": true,
        "start_tls": false
    }
]
```

Returns an empty array if no LDAP servers are configured.

You must have the `directory_service:edit` permission to view all fields; otherwise, only the display name of the directory server is returned.

For information about each setting, refer to [External directory settings](#) on page 282.

For errors, refer to [RBAC service errors](#) on page 313.

Activity service API

The activity service records changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles. Use the activity service API to query event data.

- [Forming activity service API requests](#) on page 368

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 370

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 372

Use the `events` endpoints to retrieve activity service events.

Related information

[API index](#) on page 33

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.

RBAC API requests must include a URI path following the pattern:

```
https://<DNS>:4433/activity-api/<VERSION>/<ENDPOINT>
```

The variable path components derive from:

- **DNS**: Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).
- **VERSION**: Either v1 or v2, depending on the endpoint.
- **ENDPOINT**: Either `events` or `events.csv`, depending on the endpoint.

For example, you could use any of these paths to call the [GET /v1/events](#) on page 372 endpoint:

```
https://$(puppet config print server):4433/activity-api/v1/events
https://localhost:4433/activity-api/v1/events
https://puppet.example.dns:4433/activity-api/v1/events
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Token authentication

You can use token or certificate authentication with the activity service API.

For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

To use a token in an request, you can use `puppet-access show`, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/activity-api/v1/events"
curl --header "$auth_header" "$uri"
```

Or you can use the actual token, such as:

```
auth_header="X-Authentication: <TOKEN>
uri="https://$(puppet config print server):4433/activity-api/v1/events"
curl --header "$auth_header" "$uri"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Allowed certificate authentication

You can authenticate requests with a certificate listed in RBAC's certificate allowlist, which is located at:

```
/etc/puppetlabs/console-services/rbac-certificate-allowlist
```

Important: If you edit the `rbcac-certificate-allowlist` file, you must reload the `pe-console-services` service for your changes to take effect. To reload the service run: `sudo service pe-console-services reload`

To use a certificate in a curl command, include the allowed certificate name (which must match a name in the `rbcac-certificate-allowlist` file) and, if necessary, the private key. For example:

```
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"
curl --cert "$cert" --cacert "$cacert" --key "$key" "$uri"
```

Tip: You do not need to use an agent certificate for authentication. You can use the `puppet cert generate` command to create a certificate to use specifically with the activity service API.

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they 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.

User and authentication token events

In the PE console, you can view records related to local and remote users on the **Activity** tab of the user's page. Remote user pages only show the **Role membership** and **Revocation** events. All user pages can show authentication token events.

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 or removed from a role. The display name and user ID of the affected user are displayed. These events are also shown on the Activities tab of the role's page.	User Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba) added to role Operators.
Authentication	The user logged 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 to reset the user's 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 changed their password with a password reset token.	Password reset for user Jean Jackson (973c0cee-5ed3-11e4-aa15-123b93f75cba).
Revocation	A user is revoked or reinstated.	User revoked.

The user page also reports these authentication token events:

Event	Description	Example
Creation	A token is generated for the user. The Creation event appears on the page of the user who owns the token.	Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c) generated an authentication token.

Event	Description	Example
Direct revocation	An individual token was revoked. This event appears on the page of the user who requested the revocation, not the user whose token was revoked.	Administrator (42bf351cf9ec-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	Revoked all tokens belonging to a specific user name. This event appears on the page of the user who requested the revocation, not the user whose token was revoked.	Administrator (42bf351cf9ec-40af-84ad-e976fec7f4bd) revoked all authentication tokens belonging to Amari Perez (c84bae61-f668-4a18-9a4a-5e33a97b716c).

Directory user group events

These events are listed in the console on the **Activity** tab of the user group's page.

Event	Description	Example
Importation	A directory group is imported. The initial value for each metadata field is reported (these cannot be updated in the console).	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 role's page. The group's display name and ID are provided.	Group Engineers (7dee3acc-5ed4-11e4-aa15-123b93f75cba) added to role Operators.

User role events

These events are listed in the console on the **Activity** tab of the role's page.

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 user's or group's page.	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.

The activity service also records a **Delete** event when a role is removed. However, information about **Delete** events are only available through the activity service API [Events endpoints](#) on page 372.

Orchestrator events

These events are listed in the console on the **Activity** tab of the node's page.

Event	Description	Example
Agent runs	Puppet ran as part of an orchestration job. This includes Puppet runs started from the orchestrator or the PE console.	Request Puppet agent run on node.example.com via orchestrator job 12.
Task runs	Tasks ran as part of orchestration jobs that were set up in the console or on the command line.	Request echo task on neptune.example.com via orchestrator job 9,607

Directory service settings events

These events are not exposed in the console. You must use the activity service API [Events endpoints](#) on page 372 to get information about these events.

Event	Description	Example
Update settings (except password)	A setting changed in the directory service settings, other than the password.	User rdn set to "ou=users".
Update directory service password	The directory service password changed.	Password updated.

Events endpoints

Use the events endpoints to retrieve activity service events.

GET /v1/events

Fetch information about events the activity service tracks. Web session authentication is required.

Request format

When [Forming activity service API requests](#) on page 368 to this endpoint, the request is a basic call with authentication and one or more query parameters, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/activity-api/v1/events?
service_id=classifier"

curl --header "$auth_header" "$uri"
```

The following parameters are supported. You must append the `service_id` parameter. Other parameters are optional or conditionally required.

Parameter	Value
<code>service_id</code>	Required. The ID of the service you want to query.

Parameter	Value
subject_type	Limit the activity query to a specific subject type (which is the actor of the activity). Use subject_id to further limit the query to specific IDs within the specified type. For example, you can query all user activities or specific users' activities.
subject_id	A comma-separated list of IDs associated with the defined subject type. Optional, but, if supplied, then subject_type is required.
object_type	Limit the activity query to a specific object type (which is the target of activities). Use object_id to further limit the query to specific IDs within the specified type.
object_id	A comma-separated list of IDs associated with the defined object type. Optional, but, if supplied, then object_type is required.
offset	Specify a zero-indexed integer to retrieve activity records starting from the offset point. If omitted, the default is 0. This parameter is useful for omitting initial, irrelevant results, such as test data.
order	Specify, as a string, whether records are returned in ascending (asc) or descending (desc) order. If omitted, the default is desc. Sorting is based on the activity record's submission time.
limit	Specify a positive integer to limit the number of user records returned. If omitted, the default is 1000 events.
after_service_commit_time	Specify a timestamp in ISO-8601 format if you want to fetch results after a specific service commit time. Optional.

Tip: For more nuanced queries and additional query parameters, use the [GET /v2/events](#) on page 376 endpoint.

Response format

The response contains a series of JSON objects representing event records. The response also reports the total-rows, which represents the total number of records matching the supplied query.

For example, this response was based on a request that queried the classifier service (service_id=classifier), and events performed by a specific user (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
}
```

As another example, this response was based on a request that queried the classifier service (`service_id=classifier`), and events that targeted a specific node group (`object_type=node_groups&object_id=2`):

```
{
  "commits": [
    {
      "object": {
        "id": "415dfsvdf-dfgd45dfg-4dsfg54d",
        "name": "Default Node Group"
      },
      "subject": {
        "id": "dfgdxfc145-545dfg54f-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

Fetch information about events the activity service tracks in a flat CSV format. Token-based authentication is required.

Request format

When [Forming activity service API requests](#) on page 368 to this endpoint, the request is a basic call with authentication and one or more query parameters, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/activity-api/v1/events.csv?
service_id=classifier&limit=50"

curl --header "$auth_header" "$uri"
```

The following parameters are supported. You must append the `service_id` parameter. Other parameters are optional or conditionally required.

Parameter	Definition
service_id	Required. The ID of the service you want to query.
subject_type	Limit the activity query to a specific subject type (which is the actor of the activity). Use <code>subject_id</code> to further limit the query to specific IDs within the specified type. For example, you can query all user activities or specific users' activities.
subject_id	A comma-separated list of IDs associated with the defined subject type. Optional, but, if supplied, then <code>subject_type</code> is required.
object_type	Limit the activity query to a specific object type (which is the target of activities). Use <code>object_id</code> to further limit the query to specific IDs within the specified type.
object_id	A comma-separated list of IDs associated with the defined object type. Optional, but, if supplied, then <code>object_type</code> is required.
offset	Specify a zero-indexed integer to retrieve activity records starting from the offset point. If omitted, the default is 0. This parameter is useful for omitting initial, irrelevant results, such as test data.
order	Specify, as a string, whether records are returned in ascending (<code>asc</code>) or descending (<code>desc</code>) order. If omitted, the default is <code>desc</code> . Sorting is based on the activity record's submission time.
limit	Specify a positive integer to limit the number of user records returned. If omitted, the default is 1000 events.

Tip: For more nuanced queries and additional query parameters, use the [GET /v2/events.csv](#) on page 380 endpoint.

Response format

The response contains all returned records in a flat CSV format. For example:

```
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_desc-
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_envi-
the environment to "production""
```

```

YYYY-MM-DD
18:52:02.391,users,6868e4af-2996-46c6-8e42-1ae873f8a0ba,kai.evens,node_groups,b55c209d-e68f-4096-9a2c-5ae52dd2500c,web_servers,edit,node_group_environment_override,edit_node_group_environment_override 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,change 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_enterprise::profile::orchestrator class
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_enterprise::profile::orchestrator class
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 information about events the activity service tracks. Allows filtering through query parameters and supports multiple objects for filtering results. Requires token-based authentication.

Request format

When [Forming activity service API requests](#) on page 368 to the /v2/events endpoint, you can provide multiple optional parameters for filtering results. Parameters are either appended to the URI path or supplied in a JSON body.

You can append the following parameters to the URI path:

Parameter	Definition
service_id	The ID of the service you want to query. If omitted, all services are queried.
offset	Specify a zero-indexed integer to retrieve activity records starting from the offset point. If omitted, the default is 0. This parameter is useful for omitting initial, irrelevant results, such as test data.
order	Specify, as a string, whether records are returned in ascending (asc) or descending (desc) order. If omitted, the default is desc. Sorting is based on the activity record's submission time.
limit	Specify a positive integer to limit the number of user records returned. If omitted, the default is 1000 events.

You can append additional parameters in the JSON-formatted query array. If you use the query array, you must append all parameters with --data-urlencode, instead of appending them to the URL. Each item in the query array is an object consisting of a single parameter and a value, or a pair of related parameters and values. Some parameters can be repeated to specify multiple values in the same category. For example:

```
--data-urlencode '{"query": [{"object_id": "3", "object_type": "users"}, \
                           {"subject_type": "node_groups"}, \
                           {"subject_type": "roles"}], \
```

```
{ "start": "2019-11-01T21:32:39Z", "end":  
"2019-12-01T00:00:00Z" } ] }
```

Parameters you can use in the `query` array include:

Parameter	Definition
<code>subject_id</code>	<p>Required. Limit the query to the subject (a user) with the specified ID. If <code>subject_type</code> is omitted, the type is assumed to be <code>users</code>. Currently, <code>users</code> is the only available <code>subject_type</code>.</p> <p>Place <code>subject_id</code> and <code>subject_type</code> within the same object, separated by a comma.</p>
<code>subject_type</code>	<p>Optional, but if included, you must also include <code>subject_id</code>. Refer to <code>subject_id</code> for more information.</p>
<code>object_type</code>	<p>Limit the activity query to a specific object type (which is the target of activities).</p> <p>Use <code>object_id</code> to further limit the query to a specific ID within the specified type.</p> <p>Place <code>object_type</code> within an object. If you also specify <code>object_id</code>, place it within the same object, separated by a comma.</p>
<code>object_id</code>	<p>An ID associated with a defined object type. If supplied, then <code>object_type</code> is required.</p> <p>Place <code>object_id</code> and <code>object_type</code> within the same object, separated by a comma.</p>
<code>ip_address</code>	<p>Specifies an IP address associated with activities. Supports partial string matching.</p>
<code>start</code>	<p>Supply a timestamp in ISO-8601 format. Must be used with <code>end</code> to fetch results within a specific service commit time range.</p> <p>Place <code>start</code> and <code>end</code> within the same object, separated by a comma.</p> <p>Tip: Whereas other parameters use <code>or</code> logic, the timestamp parameters use <code>and</code> logic.</p>

Parameter	Definition
end	<p>Supply a timestamp in ISO-8601 format. Must be used with <code>start</code> to fetch results within a specific service commit time range.</p> <p>Place <code>start</code> and <code>end</code> within the same object, separated by a comma.</p> <p>Tip: Whereas other parameters use <code>or</code> logic, the timestamp parameters use <code>and</code> logic.</p>

For example, the following request returns 10 classifier events performed by two specific users from 01 November 2019 through 01 December 2019:

```
curl -k -X GET -H "X-Authentication: $(puppet-access show)" \
-G "https://$(puppet config print server):4433/activity-api/v2/events" \
--data-urlencode 'service_id=classifier' \
--data-urlencode 'limit=10' \
--data-urlencode 'query=[{"object_id": "db2caca1-d6a4-4145-8240-9de9b4e654d1", "object_type": "users"}, \
{"subject_id": "db2caca1-d6a4-4145-8240-9de9b4e654d1"}, \
{"object_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8", \
"object_type": "users"}, \
{"subject_id": "5d5ab481-7614-4324-bfea-e9eeb0b22ce8"}, \
{"start": "2019-11-01T21:32:39Z", "end": "2019-12-01T00:00:00Z"}]'
```

Tip: If you supply the JSON-formatted `query` array, make sure your request uses `-G`, `--data-urlencode`, and other such valid arguments to allow the GET request to convey the JSON content.

Response format

The response contains a series of JSON objects representing event records, as well as pagination information based on the submitted query.

For example, a request to the classifier service (`service_id=classifier`) about actions performed on a specific node group (`query=[{"object_id": "415", "object_type": "node_group" }]`) might produce a response similar to:

```
{
  "commits": [
    {
      "objects": [
        {
          "id": "415dfsdf-dfgd45dfg-4dsfg54d",
          "name": "Default Node Group"
        }
      ],
      "subject": {
        "id": "dfgdgc145-545dfg54f-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" }
}

```

Responses containing information about orchestrator events, including Puppet agent runs and task runs, can return these keys:

- `start_timestamp`: A timestamp in ISO-8601 format reporting the job start time.
- `finish_timestamp`: A timestamp in ISO-8601 format reporting the job end time.
- `duration`: The job's elapsed run time, in seconds.
- `state`: One of `new`, `ready`, `running`, `stopping`, `stopped`, `finished`, or `failed`.

Failed and in-progress jobs do not return the `finish_timestamp` or `duration` keys.

For example, this partial response contains information about one commit for a Puppet run:

```

{
  "objects": [
    {
      "id": "example.delivery.puppetlabs.net",
      "name": "example.delivery.puppetlabs.net",
      "type": "node"
    }
  ],
  "subject": {
    "id": "11bf351c-f1ec-11af-11ad-e111feclalbd",
    "name": "admin"
  },
  "timestamp": "2022-08-08T23:18:52Z",
  "ip_address": "<IP_ADDRESS>",
  "events": [
    {
      "description": "request_puppet_agent_on_node",
      "finish_timestamp": "2022-08-08T23:18:52Z",
      "start_timestamp": "2022-08-08T23:18:47Z",
      "name": "puppet agent",
      "type": "puppet_agent",
      "duration": 4.898,
      "state": "finished",
      "what": "node",
      "message": "Request \"puppet agent\" run on
\"example.delivery.puppetlabs.net\" over \"pcp\" via orchestrator job
\"11\""
    }
  ]
},

```

This partial response example contains information about one commit for a task run:

```

{
  "objects": [
    {
      "id": "example.delivery.puppetlabs.net",
      "name": "example.delivery.puppetlabs.net",
      "type": "node"
    }
  ],
  "subject": {

```

```

    "id": "11bf351c-flec-11af-11ad-e111feclalbd",
    "name": "admin"
},
"timestamp": "2022-08-08T23:19:01Z",
"ip_address": "<IP_ADDRESS>",
"events": [
{
    "description": "request_facter_on_node",
    "finish_timestamp": "2022-08-08T23:19:00Z",
    "start_timestamp": "2022-08-08T23:18:58Z",
    "name": "req_facter",
    "type": "run_task",
    "duration": 2.324,
    "state": "finished",
    "what": "node",
    "message": "Request \"req_facter\" task on
\\\"example.delivery.puppetlabs.net\\\" over \"pcp\" via orchestrator job
\\\"12\\\""
}
]
},

```

Related information

[Event types reported by the activity service](#) on page 370

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.

GET /v2/events.csv

Fetch information about events the activity service tracks in a flat CSV format. Allows filtering through query parameters and supports multiple objects for filtering results. Token-based authentication is required.

Request format

Requests to the /v2/events.csv endpoint are formed in the same way as the [GET /v2/events](#) on page 376 endpoint, except that the URI path contains events.csv instead of events. For example:

```

curl -k -X GET -H "X-Authentication: $(puppet-access show)" \
-G "https://$(puppet config print server):4433/activity-api/v2/events.csv" \
--data-urlencode 'service_id=classifier' \
--data-urlencode 'query=[{"object_id": "3", "object_type": "users"}, \
{"start": "2019-11-01T21:32:39Z", "end": "2019-12-01T00:00:00Z"}]'

```

Response format

The response contains all returned records in a flat CSV format. For example:

```

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 Puppet Enterprise (PE) console has several tools you can use to monitor the current state of your infrastructure, review the results of planned or unplanned changes to your Puppet code, view reports, and investigate problems. You can find these tools under the **Enforcement** section of the console's navigation menu.

- [Monitoring infrastructure state](#) on page 381

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 387

The **Packages** page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the **Packages** page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals. Package management is available for all agent nodes.

- [Value report](#) on page 389

Value analytics give you insight into time and money saved by Puppet Enterprise (PE) automation. You can access this information on the **Value report** page in the console or through the value API.

- [Infrastructure reports](#) on page 393

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 396

The **Events** page in the console shows a summary of activity in your infrastructure. You can analyze the details of important changes, and investigate common causes behind related events. You can also examine specific class, node, and resource events, and find out what caused them to fail, change, or run as no-op.

- [Puppet Enterprise metrics and status monitoring](#) on page 399

You can use Puppet Enterprise (PE) metrics and status monitoring for your own performance tuning or provide the information to Support for troubleshooting.

- [View and manage Puppet Server metrics](#) on page 401

Puppet Server tracks performance and status metrics you can use to monitor server health and performance over time.

- [Metrics API](#) on page 410

Use the metrics API to query Java Management Extension (JMX) metrics related to Puppet Server and the orchestrator service.

- [Status API](#) on page 416

You can use the status API to check the health of Puppet Enterprise (PE) components and services. It is useful for automatically monitoring 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.

Tip: The **Status** page describes the outcome of Puppet runs on nodes in your infrastructure. This is different from the status of your overall Puppet Enterprise (PE) installation. To understand the status of your installation as a whole, use [Puppet Enterprise metrics and status monitoring](#) on page 399.

Node run statuses

The **Status** page displays each node's run status for the most recent Puppet run. Possible statuses depend on the Puppet run mode.

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 that is 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 Puppet corrected those inconsistencies to match the catalog.

Corrective change reporting is available only on agent nodes running Puppet Enterprise (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, catalog changes were successfully applied to the node.



Unchanged

This node's last Puppet run was successful, and the node was fully compliant. No changes were necessary.

Nodes run in no-op mode

No-op mode simulates a Puppet run without making changes. 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 no-op Puppet run failed or Puppet encountered an error that prevented it from simulating changes.



Would have corrective changes

During the last no-op Puppet run, Puppet found inconsistencies between the last applied catalog and this node's configuration, and, in a true run, Puppet would correct those inconsistencies to match the catalog.



Would have intentional changes

If the last no-op Puppet run had been a true run, catalog changes would have been applied to the node.



Would be unchanged

This node's last no-op Puppet run was successful, and the node was fully compliant. In a true run, 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 run status table shows the timestamp for the node's last known Puppet run and whether the node's last known run was in no-op mode. Correct the problem to resume Puppet runs on the node.

The default cutoff time for considering a node unresponsive is one hour, but you can change this with the `puppet_enterprise::console_services::no_longer_reporting_cutoff` parameter. Go to [Configure the PE console and console-services](#) on page 230 for more information.



Have no reports

Although Puppet Server is aware the node exists, the node has never submitted a Puppet report because the node is a new node, the node has never come online, or the node's copy of Puppet is not configured correctly.

Expired or deactivated nodes are shown on the **Status** page for seven days. To extend the amount of time that you can view or search these nodes, change the `node-ttl` setting in PuppetDB. Changing this setting impacts resources and exported resources.

Special categories

In addition to reporting each node's run status, the **Status** page provides a secondary count of nodes in special categories:

Intended catalog failed

During the last Puppet run, the intended catalog for this node failed and Puppet substituted a cached catalog, according to your configuration settings.

This typically occurs if there are compilation errors in your Puppet code. Check the Puppet run log for details.

This category is shown only if one or more agents failed to retrieve a valid catalog from Puppet Server.

Enforced resources found

During the last no-op Puppet run, one or more resources were enforced, according to your use of the `noop => false` metaparameter setting.

This category is shown only if enforced resources are present on at least one node.

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 have the highest alert level, and *no change* events have the lowest alert level.

Run status icon	Definitely happened	Might also have happened
	Failure	Corrective change, intentional change, no change
	Corrective change	Intentional change, no change

Run status icon	Definitely happened	Might also have happened
	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, because this status has the highest alert level of all statuses that occurred during the run.

[Node run statuses](#) on page 382 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. If the no-op flags on a node's resources are all set to false, then 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 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>

Monitor PE services

You can monitor the status of core services in the Puppet Enterprise (PE) has console or with the `puppet infrastructure status` command.

Component or service	Console status monitor	Command line status
Activity service	Yes	Yes
Agentless Catalog Executor (ACE) service	No	Yes
Bolt service	No	Yes
Classifier service	Yes	Yes
Code Manager service	Yes	Yes
Orchestrator service	Yes	Yes
Puppet Communications Protocol (PCP) broker	No	Yes
PostgreSQL	No	Yes
Puppet Server	Yes	Yes
PuppetDB	Yes	Yes
Role-based access control (RBAC) service	Yes	Yes

View the Puppet Services status monitor

The **Puppet Services status** monitor provides a visual overview of the current state of core services. You can use it to quickly determine whether an unresponsive or restarting service is causing an issue with your deployment.

1. In the console, click **Status**.
2. Click **Puppet Services status** to open the monitor.

A check mark is shown next to **Puppet Services status** if all applicable services are accepting requests. If no data is available, a question mark is shown. If any services are restarting or not accepting requests, a warning icon is shown.

puppet infrastructure status command

The `puppet infrastructure status` command returns errors and alerts from Puppet Enterprise (PE) components and services.

The command reports separately on the primary server and any compilers or replicas in your environment. You must run the command as root.

Viewing and managing packages

The **Packages** page in the console shows all packages in use across your infrastructure by name, version, and provider, as well as the number of instances of each package version in your infrastructure. Use the **Packages** page to quickly identify which nodes are impacted by packages you know are eligible for maintenance updates, security patches, and license renewals. Package management is available for all agent nodes.

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. Package inventory reporting is available on nodes with Puppet agent 1.6.0 or later installed, including nodes that don't have an active configuration on the Puppet Server.

1. In the console, click **Node groups**.
 - If you want to collect package data on all your nodes, click the **PE Agent** node group.
 - If you want to collect package data on a subset of your nodes, click **Add group** and create a new classification node group. Select **PE Agent** as the group's parent name. After the new node group is set up, use the **Rules** tab to dynamically add the relevant nodes.
2. Click **Classes**. In the **Add new class** field, select `puppet_enterprise::profile::agent` and click **Add class**.
3. In the `puppet_enterprise::profile::agent` class, set the **Parameter** to `package_inventory_enabled` and the **Value** to `true`. Click **Add parameter**, and commit changes.
4. Run Puppet to apply these changes to the nodes in your node group.

Puppet enables package inventory collection on this Puppet run, and begins collecting package data and reporting it on the **Packages** page on each subsequent Puppet run.

5. Run Puppet a second time to begin collecting package data, then click **Packages**.

View and manage package inventory

To view and manage the complete inventory of packages on your systems, use the **Packages** page in the console.

Before you begin

Make sure you have enabled package data collection for the nodes you wish to view.

Tip: If all the nodes on which a certain package is installed are deactivated, but the nodes' specified node-purge-ttl period has not yet elapsed, instances of the package still appear in summary counts on the **Packages** page. To correct this issue, adjust the node-purge-ttl setting and run garbage collection.

1. Run Puppet to collect the latest package data from your nodes.
2. In the console, click **Packages** to view your package inventory. To narrow the list of packages, enter the name or partial name of a package in the **Filter by package name** field and click **Apply**.
3. Click any package name or version to enter the detail page for that package.
4. On a package's detail page, use the **Version** selector to locate nodes with a particular package version installed.
5. Use the **Instances** selector to locate nodes where the package is not managed with Puppet, or to view nodes on which a package instance is managed with Puppet.

To quickly find the place in your manifest where a Puppet-managed package is declared, select a code path in the **Instances** selector and click **Copy path**.

6. To modify a package on a group of nodes:
 - If the package is managed with Puppet, select a code path in the **Instances** selector and click **Copy path**, then navigate to and update the manifest.
 - If the package is not managed with Puppet, click **Run > Task** and create a new task.

View package data collection metadata

The `puppet_inventory_metadata` fact reports whether package data collection is enabled on a node, and it 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 want 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` and 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 the 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 saved by Puppet Enterprise (PE) automation. You can access this information on the **Value report** page in the console or through the value API.

The information in the value report and the value API provide details about automated changes PE makes to nodes and estimates time saved by each type of change based on intelligent defaults or values you provide. If you specify an average hourly salary, the report also estimates cost savings from automated changes.

To ensure your value analysis is accurate:

- Make sure tasks, plans, and Puppet runs are processing normally because value analysis doesn't track failed runs.
- If you use the value API, query the endpoint regularly to gather data over time.

Value report defaults

Value analytics uses intelligent defaults to estimate time freed by automated changes. These defaults are based on customer research and take into account time to triage, research, and fix issues, as well as context switching. You can also provide your own default values.

When you query the value API, you can specify low, med, or high estimates for time freed parameters, or provide an exact value in minutes based on averages in your organization. In the console **Value report**, you specify an exact value in minutes. Unless you specify otherwise, both the API and the console use the med values. The baseline intelligent default values are as follows:

Parameter	Intelligent default values
minutesFreedPerCorrectiveChange	<ul style="list-style-type: none"> • low: 90 minutes • med: 180 minutes • high: 360 minutes
minutesFreedPerIntentionalChange	<ul style="list-style-type: none"> • low: 30 min • med: 90 minutes • high: 180 minutes
minutesFreedPerTaskRun	<ul style="list-style-type: none"> • low: 30 minutes • med: 90 minutes • high: 180 minutes
minutesFreedPerPlanRun	<ul style="list-style-type: none"> • low: 90 minutes • med: 180 minutes • high: 360 minutes

You can change the low, med, and high times by specifying any of the `value_report_*` parameters in the PE Console node group in the `puppet_enterprise::profile::console` class.

GET /api/reports/value

Use the GET /api/reports/value endpoint to retrieve information about time and money freed by Puppet Enterprise (PE) automation.

Request format

You must provide well-formed HTTP(S) requests. By default, the value API uses the standard HTTPS port for console communication, which is port 443. You can omit the port from your requests unless you want to specify a different port.

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.

This is an example of a basic, authenticated value API request without any parameters:

```
curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: <RBAC_TOKEN>" \
"https://<HOSTNAME>/api/reports/value"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

You can use these parameters, attached with --data-urlencode, to modify your value API requests:

Parameter	Description	Default value
averageHourlySalary	Numeric value specifying average hourly salary to use to cost savings for automated work.	None
startDate	A date in yyyy-mm-dd format.	Today less nine days (One week plus two days in the past)
endDate	A date in yyyy-mm-dd format. If you specify today's date, the response contains provisional data.	today less two days
minutesFreedPerCorrectiveChange	high, or any numeric value in minutes.	med
minutesFreedPerIntentionalChange	high, or any numeric value in minutes.	med
minutesFreedPerTaskRun	low, med, high, or any numeric value in minutes.	med
minutesFreedPerPlanRun	low, med, high, or any numeric value in minutes.	med

The numerical basis for minutesFreed parameters are controlled by [Value report defaults](#) on page 389.

Response format

The response is a JSON object listing details about time and cost freed. Responses use these keys:

Key	Definition
startDate	Start date for the reporting period.
endDate	End date for the reporting period.

Key	Definition
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 <code>minutesFreedPerCorrectiveChange</code> 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 <code>minutesFreedPerIntentionalChange</code> query parameter.
totalNodesAffectedByTaskRuns	Total number of nodes affected by successful task runs during the reporting period.
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 <code>minutesFreedPerTaskRun</code> 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 <code>minutesFreedPerPlanRun</code> query parameter.
totalMinutesFreed	Total number of minutes free by all automated changes.
totalDollarsSaved	If the query specified an <code>averageHourlySalary</code> , total cost savings for all automated changes.

Request and response examples

This request generates a report for specified dates using the default time freed values:

```
curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: <RBAC_TOKEN>" \
-G "https://<HOSTNAME>/api/reports/value" \
--data-urlencode 'startDate=2020-07-08' \
--data-urlencode 'endDate=2020-07-15'
```

The result is:

```
{
  "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
}

```

This request generates cost savings using default report dates and time freed values:

```

curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: <rbac token>" \
-G "https://<pe-console-fqdn>/api/reports/value" \
--data-urlencode 'averageHourlySalary=40'

```

The result is:

```

{
  "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,
}

```

This request generates a report with custom values for time freed:

```

curl -X GET --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" \
-H "X-Authentication: $(cat ~/.puppetlabs/token)" \
-G "https://<pe-console-fqdn>/api/reports/value" \
--data-urlencode 'minutesFreedPerCorrectiveChange=10' \
--data-urlencode 'minutesFreedPerIntentionalChange=20' \
--data-urlencode 'minutesFreedPerTaskRun=30' \
--data-urlencode 'minutesFreedPerPlanRun=40'

```

The result is:

```

{
  "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	Number of resources that were supposed to restart but didn't. For example, if changes to one resource notify another resource to restart, and that resource doesn't restart, a failed restart is reported. It's an indirect failure that occurred in a resource that was otherwise unchanged.

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>-<timestampl>`.

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>-<timestampl>`.

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.

Metric	Description
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.
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`.

Puppet Enterprise metrics and status monitoring

You can use Puppet Enterprise (PE) metrics and status monitoring for your own performance tuning or provide the information to Support for troubleshooting.

There `puppet_metrics_collector` and `pe_status_check` modules are bundled with PE. These modules help you track the status of your PE installation as a whole.

Tip: The information reported by these modules is different from information presented in [Infrastructure reports](#) on page 393 and [Node run statuses](#) on page 382, which report on the outcome of Puppet runs.

You can also use APIs or our Splunk plugin to [View and manage Puppet Server metrics](#) on page 401.

About the `puppet_metrics_collector` module

The `puppet_metrics_collector` module collects metrics from the status endpoints of Puppet Enterprise (PE) services.

The `puppet_metrics_collector` module is installed with PE and is partially enabled by default.

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

The following two parameters control metrics collection:

`puppet_enterprise::enable_metrics_collection`

A Boolean specifying whether the primary server collects metrics from PE services, such as Puppet Server and PuppetDB .

Default: `true`

`puppet_enterprise::enable_system_metrics_collection`

A Boolean specifying whether your infrastructure nodes collect metrics from the operating system your PE services run on. To allow the collection of system metrics, `sysstat` must be installed and enabled on your operating system.

Default: `false`

Visit the [puppet_metrics_collector Forge page](#) to learn about this module's other classes and features, such as retention time, collection frequency, and parameters for specific services.

You can use the [puppet_operational_dashboards module](#) to view PE metrics.

Tip: You can also use APIs or our Splunk plugin to [View and manage Puppet Server metrics](#) on page 401.

Enable or disable metrics collection

The `puppet_metrics_collector` module is partially enabled by default, and you can manually configure the metrics collection parameters.

The metrics collection parameters accept Boolean values. Setting both parameters to `false` disables the module entirely.

1. In the PE console, click **Node groups** and select the **PE Infrastructure** node group.
2. Configure the following parameters according to your requirements:
 - `puppet_enterprise::enable_metrics_collection`: Set this parameter on the **Configuration data** tab to determine whether your primary server collects metrics for PE services.
 - `puppet_enterprise::enable_system_metrics_collection`: Set this parameter on the **Classes** tab to determine whether your infrastructure nodes collect metrics from the operating system your PE services run on.

Note: To allow the collection of system metrics, `sysstat` must be installed and enabled on your operating system.

3. Commit your changes and run Puppet.

Related information

[How to configure PE](#) on page 211

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. 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.

About the `pe_status_check` module

The `pe_status_check` module can alert you when your Puppet Enterprise (PE) installation is not in an ideal state, based on preset indicators, and describe how you can resolve or improve the detected issue.

Important: If you have previously specified a version of this module, from the Forge or other sources, in your code, we recommend removing this version to allow the version bundled with PE to be asserted.

By default, the `pe_status_check` module uses the `pe_status_check` fact to compare your installation to a predetermined ideal state. This fact collects information about your PE infrastructure components. You can optionally enable the `agent_status_check` fact to collect information about agent nodes that are not infrastructure nodes. To enable this fact, classify `pe_status_check::agent_status_enable` to your nodes.

The `pe_status_check` module produces reports based on the information collected by the `pe_status_check` fact (and the `agent_status_check` fact, if it is enabled). The module recommends remedial actions you can take to either resolve a deficiency or improve your installation's tuning.

To get reports from the module, you can:

- Run the `pe_status_check::infra_summary` and `pe_status_check::agent_summary` plans. These are [setup requirements](#) and various parameters and commands you can use for [running the plans](#).
- [Use a Puppet Query Language \(PQL\) query](#).

To enable notifications in reports, you must [declare the `pe_status_check` class](#).

These sections of the module's Forge page explain how to interpret the module's reports and the information that the facts collect:

- [Fact: `pe_status_check`](#)
- [Fact: `agent_status_check`](#)

View and manage Puppet Server metrics

Puppet Server tracks performance and status metrics you can use to monitor server health and performance over time.

You can retrieve, track, and visualize Puppet Server metrics with:

- The [Puppet Report Viewer app](#) for Splunk. You'll need the `splunk_hec` and `pe_event_forwarding` modules along with the add-on. For information about this option, refer to these blog posts:
 - [PE Metrics in Splunk: Puppet Report Viewer 3.1](#)
 - [Introducing Puppet and Splunk integrations to improve reporting speed and scale](#)
- The `puppet_operational_dashboards` module (which you can use along with the `puppet_metrics_collector` module module that is already bundled with PE).
- The [Metrics API](#) on page 410 and [Status API](#) on page 416 endpoints.
- Customizable, networked Graphite and Grafana instances. While the `grafanadash` and `puppet-graphite` modules are not Puppet-supported modules (they are provided for testing and demonstration purposes only), you can learn about these options in [Get started with Graphite](#) on page 401 and [Available Graphite metrics](#) on page 406.
- [Get started with Graphite](#) on page 401

[Graphite](#) is a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. Puppet Enterprise (PE) can export many metrics to Graphite. After enabling Graphite support, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

- [Available Graphite metrics](#) on page 406

These HTTP and Puppet profiler metrics are available from the Puppet Server and can be added to your metrics reporting.

Get started with Graphite

[Graphite](#) is a third-party monitoring application that stores real-time metrics and provides customizable ways to view them. Puppet Enterprise (PE) can export many metrics to Graphite. After enabling Graphite support, Puppet Server exports a set of metrics by default that is designed to be immediately useful to Puppet administrators.

Restriction: Graphite setups are deeply customizable and can report many different Puppet Server metrics on demand; however, this requires considerable configuration and additional server resources. Furthermore, the `grafanadash` and `puppet-graphite` modules are not Puppet-supported.

We recommend using another method to [View and manage Puppet Server metrics](#) on page 401, such as the `puppet_operational_dashboards` module, our Splunk plugin, or the Metrics API.

To use Graphite with PE, you must:

- [Install and configure a Graphite server](#).
- [Enable Puppet Server's Graphite support](#) on page 402.
- (Optional) Use the [Grafana](#) dashboard extension for Graphite to visualize metrics. To see a demonstration of this setup, [Use the grafanadash module](#) on page 401.

Use the `grafanadash` module

[Grafana](#) provides a web-based, customizable, Graphite-compatible dashboard. The `grafanadash` module installs and configures a basic Graphite test instance with the Grafana extension. When installed on a Puppet agent, the purpose of this module is to demonstrate how Graphite and Grafana can consume and display Puppet Server metrics.



CAUTION: The [grafanadash module](#) is not a Puppet-supported module. It is for testing and demonstration purposes only, is considered insecure, and is tested against CentOS 7 only. Install this module only on a dedicated agent. **Do not install the grafanadash module on your primary server.** This module makes the following 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 `grafanadash` 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.

For the above reasons, we recommend using another method to [View and manage Puppet Server metrics](#) on page 401, such as the `puppet_operational_dashboards` module or the Metrics API.

Install the `grafanadash` module

Install the `grafanadash` module on a dedicated *nix agent. The module's `grafanadash::dev` class installs and configures a Graphite server, the Grafana extension, and a default dashboard.

1. Install a dedicated *nix PE agent to serve as the Graphite server. For instructions, refer to [Installing agents](#) on page 145.
2. As root on the agent node, run: `sudo puppet module install puppetlabs-grafanadash`
3. As root on the agent node, run: `sudo puppet apply -e 'include grafanadash::dev'`

Run Grafana

Grafana runs as a web-based dashboard, and the `grafanadash` module configures it to use port 10000 by default. To view Puppet Server metrics in Grafana, you must configure a metrics dashboard.

Grafana does not display Puppet metrics displayed by default. You must create a metrics dashboard or edit and import a JSON-based dashboard, such as our [sample metrics dashboard JSON](#) file.

Tip: You can also use the [puppet_operational_dashboards](#) module to visualize Puppet Server metrics.

1. Open a web browser on a computer that can reach your `grafanadash` agent node and navigate to `http://<AGENT_HOSTNAME>:10000`. You'll see a test screen indicating whether Grafana can successfully connect to your Graphite server. If Grafana is configured to use a hostname that your current computer can't resolve, click **View details** and go to the **Requests** tab to determine the hostname Grafana is trying to use. Then add the IP address and hostname to the `hosts` file.
 - On *nix and macOS agents, the file is located at: `/etc/hosts`
 - On Windows agents, the file is located at: `C:\Windows\system32\drivers\etc\hosts`
2. Download the [sample metrics dashboard JSON](#) file, save the file as `sample_metrics_dashboard.json`, and open it in a text editor on the same computer you're using to access Grafana.
3. Throughout the file, replace `primary.example.com` with the hostname of your primary server.

Important: The hostname value **must** also be used as the `metrics_server_id` value when you [Enable Puppet Server's Graphite support](#) on page 402.

4. Save the file.
5. In the Grafana UI, click **Search (Folder icon) > Import > Browse**, then select your `sample_metrics_dashboard.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 support](#) on page 402. For information about the aspects of the sample dashboard, refer to [Sample Grafana dashboard graphs](#) on page 403.

Enable Puppet Server's Graphite support

Use the **PE Master** node group in the Puppet Enterprise (PE) console to configure Puppet Server's metrics output settings.

1. In the PE console, go to **Node groups > PE Infrastructure > PE Master**.

2. On the **Classes** tab, locate the `puppet_enterprise::profile::master` class, and add these parameters:
 - a) Set `metrics_graphite_enabled` to `true` (the default is `false`).
 - b) Set `metrics_server_id` to the primary server hostname.
 - c) Set `metrics_graphite_host` to the hostname of the agent node where you're running Graphite and Grafana.
 - d) Set `metrics_graphite_update_interval_seconds` to an integer representing a number of seconds. This is the frequency at which Graphite updates, and the default value is 60 seconds.
3. Verify that these parameters are set to their default values, unless your Graphite server uses a non-standard port:
 - a) Confirm `metrics_jmx_enabled` is set to `true`.
 - b) Confirm `metrics_graphite_port` is set to 2003 or the Graphite port on your Graphite server.
 - c) Confirm `profiler_enabled` is set to `true`.
4. Commit changes.

Sample Grafana dashboard graphs

In the [Run Grafana](#) on page 402 steps, you used a JSON file to set up a sample Grafana dashboard. You can customize this dashboard by clicking the title of any graph and clicking **Edit**.

Graph name	Description
Active requests	<p>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.</p> <p>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.</p>
Request durations	<p>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.</p> <p>A sharp upward curve for all request types indicates an overloaded server. Expect these to trend downward after the server load is reduced.</p>
Request ratios	<p>This graph shows how many requests of each type that Puppet Server has handled. Under normal circumstances, you'll see about the same number of catalog, node, or report requests, because these all happen once per agent run. The number of file and file metadata requests correlate to how many remote file resources are in the agents' catalogs.</p>
External HTTP Communications	<p>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.</p>

Graph name	Description
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.
	<p>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.</p> <p>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.</p> <p>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 JRuby max active instances on page 206 to allow it to allocate more JRubies. Otherwise, consider adding additional compilers to distribute the catalog compilation load.</p>

Graph name	Description
JRuby Timers	<p>This graph tracks these JRuby pool metrics:</p> <ul style="list-style-type: none"> Borrow time: The mean amount of time that Puppet Server uses (or "borrows") each JRuby from the pool. Wait time: The total amount of time that Puppet Server waits for a free JRuby instance. Lock held time: 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. Lock wait time: 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. It demonstrates:

- The full names of Puppet's exported Graphite metrics
- A way to add targets directly to an exported Grafana dashboard's JSON content

This example assumes the Puppet Server instance has a domain of `primary.example.com`.

```

"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"
    }
  ]
}

```

Refer to the complete [Grafana dashboard JSON sample](#) file for a complete, 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 example, 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 example, the `puppetlabs.<PRIMARY_HOSTNAME>.compiler.mean` metric returns only the mean length of time it takes Puppet Server to compile a catalog.

To organize this reference, we've separated the metrics 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.

Restriction: 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. The default exported metrics are listed below. To export additional metrics, you can [Modify exported metrics](#) on page 410.

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 [Modify exported metrics](#) on page 410.
 - `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_metadata-/*/-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_metadata-/*/-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`

Modify exported metrics

In addition to the default metrics, you can also export metrics measuring specific environments and nodes managed by Puppet Server.

The `puppet_enterprise::profile::master::metrics_puppetserver_metrics_allowed` parameter 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>`, which represents 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. For example:

- To track the compilation time for the production environment, add `compiler.compile.production` to the `metrics-allowed` list.
- To track only the `my.node.localdomain` node in the production environment, add `compiler.compile.production.my.node.localdomain` to the `metrics-allowed` list.

Metrics API

Use the metrics API to query Java Management Extension (JMX) metrics related to Puppet Server and the orchestrator service.

Tip: You can use the [GET /status/v1/services/<SERVICE NAME>](#) on page 422 endpoint to get a summary of metrics. Form your request as: `/status/v1/services/orchestrator-service?level=debug`

There are many metrics available. For example, the "depoy-queue.length" metric reports how many nodes are waiting in queue to execute a deployment due to the `global-concurrent-compiles` setting, and the "task-queue.length" metric reports how many nodes are waiting in queue to execute a task due to the `task-concurrency` setting. The "jobs-created" metric expresses how many jobs were created in the current instance, and the "puppet-run-time" metric describes a trailing five-minute average of how long it takes a Puppet run to complete (only for Puppet runs triggered by the orchestrator).

- [Metrics API v2](#) on page 411

The `/metrics/v2/` endpoints use the Jolokia library for Java Management Extension (JMX) metrics to query Orchestrator service metrics.

- [Metrics API v1](#) on page 414

Puppet Enterprise (PE) includes an optional web endpoint for Java Management Extension (JMX) metrics managed beans (MBeans).

Related information

[API index](#) on page 33

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

Metrics API v2

The `/metrics/v2/` endpoints use the Jolokia library for Java Management Extension (JMX) metrics to query Orchestrator service metrics.

[Jolokia](#) is an extensive, open-source metrics library. We've described how to use the metrics according to the default Puppet Enterprise (PE) configuration; however, you can find more features described in the [Jolokia documentation](#).

For security reasons, by default, we only enable the read-access Jolokia interface, which includes the `read`, `list`, `version`, and `search` operations. You can [Configure Jolokia](#) on page 411 if you want to change the security access policy.

Configure Jolokia

You can customize your Jolokia security access policy and `metrics.conf` settings. You can also use these steps to disable the `/metrics/v2/` endpoints.

1. To change the security access policy:

- a) Create a `jolokia-access.xml` file at the following location:

```
/etc/puppetlabs/orchestration-services/jolokia-access.xml
```

- b) Populate the file contents according to your desired Jolokia access policy (as described in the [security chapter of the Jolokia documentation](#)), and uncomment the following parameter:

```
metrics.metrics-webService.jolokia.servlet-init-params.policyLocation
```

- c) Save the file and restart the Puppet Server service.

2. For additional configuration options, refer to the `metrics.metrics-webService.jolokia.servlet-init-params` table in the `metrics.conf` file located at:

```
/etc/puppetlabs/orchestration-services/conf.d/metrics.conf
```

[Jolokia's Servlet init parameters documentation](#) explains the various options available in this table.

Tip: To disable the `/metrics/v2/` endpoints, open the `metrics.conf` file and set the `metrics.metrics-webService.jolokia.enabled` parameter to `false`.

Forming metrics API requests

The metrics API accepts well-formed HTTPS requests.

Orchestrator API requests must include a URI path following the pattern:

```
https://<DNS>:<PORT>/metrics/v2/<OPERATION>
```

The variable path components derive from:

- DNS: Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).

- **PORT:** The PuppetDB service port.
- **OPERATION:** One or more sections specifying the operation for the request, such as `list` or `read`. Some operations require, or allow, additional modifiers such as queries, attributes, and MBean names.

For example, you could use these paths to call the [GET /metrics/v2/<OPERATION>](#) on page 413 endpoint with the `list` operation:

```
https://$(puppet config print server):8081/metrics/v2/list
https://puppet.example.dns:8081/metrics/v2/list
```

To form a complete curl command, you need to provide appropriate curl arguments, and authorization (in the form of a Puppet certificate), the content type, and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Metrics API wildcards and filtering

The `/metrics/v2/` endpoints support globbing (wildcard selection) and response filtering. You can also combine these features in the same request.

For example, this request uses [GET /metrics/v2/<OPERATION>](#) on page 413 with wildcards and filtering to get only collection counts and times from garbage collection data:

```
curl "http://puppet.example.dns:8081/metrics/v2/read/
java.lang:name=*,type=GarbageCollector/CollectionCount,CollectionTime"
```

The response is:

```
{
  "request": {
    "mbean": "java.lang:name=*,type=GarbageCollector",
    "attribute": [
      "CollectionCount",
      "CollectionTime"
    ],
    "type": "read"
  },
  "value": {
    "java.lang:name=PS Scavenge,type=GarbageCollector": {
      "CollectionTime": 1314,
      "CollectionCount": 27
    },
    "java.lang:name=PS MarkSweep,type=GarbageCollector": {
      "CollectionTime": 580,
      "CollectionCount": 5
    }
  },
  "timestamp": 1497977710,
  "status": 200
}
```

Refer to the [Jolokia protocol documentation](#) for more information.

GET /metrics/v2/<OPERATION>

Retrieve orchestrator service metrics data or metadata.

Request format

When [Forming metrics API requests](#) on page 411 to this endpoint, you must specify an operation. Some operations also require you to specify a query. For example:

```
GET /metrics/v2/<OPERATION>/<QUERY>
```

As a starting point, use the `list` operation to get a list of all MBeans:

```
GET /metrics/v2/list
```

Using information returned from the `list` operation, you can form more complex and targeted queries. For example, this request uses the `read` operation to query registered logger names:

```
GET /metrics/v2/read/java.util.logging:type=Logging/LoggerNames
```

The request format to query MBeans is:

```
GET /metrics/v2/read/<MBEAN_NAMES>/<ATTRIBUTES>/<INNER_PATH_FILTER>
```

MBean names are created by joining the first two keys in the `list` response's `value` object with a colon (which are the domain and prop `list`, in Jolokia terms), such as `java.util.logging:type=Logging`.

Attributes are derived from the `attr` object, which is within the `value` object in the `list` response.

If you specify multiple MBean names or attributes, use comma separation, such as: /
`java.lang:name=*,type=GarbageCollector`/

The inner path filter is optional and depends on the MBeans and attributes you are querying.

You must use the `read` operation to query MBeans.

Tip: Requests can also use wildcards and filtering, as described in [Forming metrics API requests](#) on page 411.

For more complex queries, or queries containing special characters, use [POST /metrics/v2/<OPERATION>](#) on page 414.

Response format

A successful request returns a JSON object containing a series of objects, arrays, and/or key-value pairs describing metrics data or metadata, based on the content of the request.

For example, the response to `GET /metrics/v2/list` contains metadata about MBeans you can use to create targeted queries, such as:

```
{
  "request": {
    "type": "list"
  },
  "value": {
    "java.util.logging": {
      "type=Logging": {
        "op": {
          "getLoggerLevel": {
            ...
          },
          ...
        }
      }
    }
  }
}
```

```

    "attr": {
      "LoggerNames": {
        "rw": false,
        "type": "[Ljava.lang.String;",
        "desc": "LoggerNames"
      },
      "ObjectName": {
        "rw": false,
        "type": "javax.management.ObjectName",
        "desc": "ObjectName"
      }
    },
    "desc": "Information on the management interface of the MBean"
  }
}
...
}
}

```

In contrast, the response to a targeted query, such as /metrics/v2/read/java.util.logging:type=Logging/LoggerNames, contains more specific data. For example:

```

{
  "request": {
    "mbean": "java.util.logging:type=Logging",
    "attribute": "LoggerNames",
    "type": "read"
  },
  "value": [
    "javax.management.snmp",
    "global",
    "javax.management.notification",
    "javax.management.modelmbean",
    "javax.management.timer",
    "javax.management",
    "javax.management.mlet",
    "javax.management.mbeanserver",
    "javax.management.snmp.daemon",
    "javax.management.relation",
    "javax.management.monitor",
    "javax.management.misc",
    ""
  ],
  "timestamp": 1497977258,
  "status": 200
}

```

POST /metrics/v2/<OPERATION>

Use more complicated queries to retrieve orchestrator service metrics data or metadata.

POST /metrics/v2/ is functionally the same as [GET /metrics/v2/<OPERATION>](#) on page 413, except that your query is appended in JSON format. This is useful when your query is complex or includes special characters.

When forming your request, the content type is `application/json` and the body must be a JSON object.

Metrics API v1

Puppet Enterprise (PE) includes an optional web endpoint for Java Management Extension (JMX) metrics managed beans (MBeans).

Restriction: The `metrics/v1/` endpoints are deprecated. We recommend using the [Metrics API v2](#) on page 411 endpoints instead.

If you choose to use the deprecated v1 endpoints, metrics are returned only when the request contains the `level=debug` parameter. The response structure might change in future versions.

The v1 endpoints include:

- GET /metrics/v1/mbeans
- POST /metrics/v1/mbeans
- GET /metrics/v1/mbeans/<name>

Set the following parameter in Hiera to enable these endpoints:

```
puppet_enterprise::master::puppetserver::metrics_webservice_enabled: true
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

For information about Java, refer to the Java documentation:

- [Java Management Extension \(JMX\)](#)
- [Managed beans \(MBeans\)](#)

GET /metrics/v1/mbeans (deprecated)

Lists available MBeans.

Response keys

The response consists of a key-value pairs where the key is the name of a valid MBean and the value is a URI you can use to request the MBean's attributes.

POST /metrics/v1/mbeans (deprecated)

Retrieves requested MBean metrics.

Request format

The request body must contain one of the following:

- A JSON object whose values are metric names
- A JSON array of metric names
- A JSON string containing a single metric's name

Use [GET /metrics/v1/mbeans \(deprecated\)](#) on page 415 to get a list of metric names.

Response format

The response is either a JSON object or array, depending on the request format:

- Requests supplying 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 supplying 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 supplying a JSON string return the a JSON object of the Mbean's attributes for the given metric name.

GET /metrics/v1/mbeans/<name> (deprecated)

Reports on a single MBean metric.

Request format

The request doesn't require any parameters, but the endpoint URI path must correspond to a metric returned by [GET /metrics/v1/mbeans \(deprecated\)](#) on page 415.

For example, this curl request queries data on MBean memory usage:

```
curl "http://localhost:8080/metrics/v1/mbeans/java.lang:type=Memory"
```

Response format

The response contains a JSON object mapping strings to values. The keys and values returned in the response depend on the metric supplied in the request.

For example, a request querying MBean memory usage (such as the `java.lang:type=Memory` metric), might return a response similar to the following:

```
{
  "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"
}
```

Status API

You can use the status API to check the health of Puppet Enterprise (PE) components and services. It is useful for automatically monitoring your infrastructure, removing unhealthy service instances from a load-balanced pool, checking configuration values, or troubleshooting issues in PE.

The status API endpoints listen on several ports. You can use the endpoints to query all services on a specified port or query an individual service on a specified port. The ports and services on each port are as follows:

Status API category	Port	Services
Console-services status API	4433	<ul style="list-style-type: none"> • RBAC • Activity service • Classifier

Status API category	Port	Services
Puppet Server status API	8140	<ul style="list-style-type: none"> Code Manager File sync client File sync storage Puppet Server PCP broker (compilers) PCP broker v2 (compilers)
Orchestrator status API	8143	<ul style="list-style-type: none"> Orchestrator PCP broker (primary server) PCP broker v2 (primary server)
PuppetDB status API	8081	PuppetDB

Important: The status API documentation uses default ports. If you changed a service's port, you might need to change the port number in your endpoint request.

Endpoint responses can return an overall health status (`healthy`, `error`, or `unknown`) and detailed information, such as database availability, the health of other required services, or connectivity to the primary server.

- [Status API authentication](#) on page 418

Token-based authentication is not required to access the status API. You can choose to authenticate requests with certificates or you can use HTTP to access the API without authentication.

- [Forming status API requests](#) on page 418

When forming status API requests, you must specify the port corresponding to the Puppet Enterprise (PE) service you want to inspect.

- [Status API: services endpoint](#) on page 419

The `/services` endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

- [Status API: services plaintext endpoint](#) on page 424

The status `service plaintext` endpoints are intended for load balancers that don't support JSON parsing or parameter setting. These endpoints return simple string bodies (either the service's state or a simple error message) and a relevant status code.

- [Status API: metrics endpoint](#) on page 425

Puppet Server can track advanced metrics to give you additional insight into its performance and health.

Related information

[API index](#) on page 33

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

Status API authentication

Token-based authentication is not required to access the status API. You can choose to authenticate requests with certificates or you can use HTTP to access the API without authentication.

Certificate authentication

You can authenticate requests with 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
```

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.

To use the certificate in a curl request, you must include the allowed certificate name (which must match a name in the `rbc-certificate-allowlist` file) and the private key. This example shows how to use `puppet` commands to include an allowed certificate in a curl request:

```
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"
```

For information about using `puppet` commands to populate curl arguments, go to [Using example commands](#) on page 28.

Tip: You do not need to use an agent certificate for authentication. You can use `puppet cert generate` to create a new certificate to use specifically with the API.

HTTP authentication

Status API endpoints can be served over HTTP, which does not require any authentication, but this is disabled by default. To enable HTTP:

1. In the PE console, go to the **PE Console** node group.
2. On the `puppet_enterprise::profile::console` class, set `console_services_plaintext_status_enabled` to true.

The default HTTP status endpoint port is 8123. To change the port:

1. In the PE console, go to the **PE Console** node group.
2. On the `puppet_enterprise::profile::console` class, set the `console_services_plaintext_status_port` parameter to the relevant port number.

Forming status API requests

When forming status API requests, you must specify the port corresponding to the Puppet Enterprise (PE) service you want to inspect.

Status API requests must include a URI path following the pattern:

```
https://<DNS>:<PORT>/status/v1/<ENDPOINT>
```

The variable path components derive from:

- DNS: Your PE console host's DNS name. You can manually enter it or use a `puppet` command, as explained in [Using example commands](#) on page 28.
- PORT: The port associated with the service(s) you want to query.
- ENDPOINT: One or more sections specifying the endpoint, such as `services` or `simple`. Some endpoints require additional sections, such as the `GET /status/v1/services/<SERVICE NAME>` on page 422 endpoint.

For example, to call the `GET /status/v1/services` on page 419 endpoint for all PE services on port 8140, you could use:

```
https://$(puppet config print server):8140/status/v1/services
```

To call the `GET /status/v1/services/<SERVICE NAME>` on page 422 endpoint for the RBAC service on port 4433, you could use either of these paths:

```
https://puppet.status.example:4433/status/v1/services/rbac-service
https://(puppet config print server):4433/status/v1/services/rbac-service
```

To form a complete curl command, you need to provide appropriate curl arguments, [Status API authentication](#) on page 418, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For information about `puppet config` commands and curl commands in Windows, go to [Using example commands](#) on page 28.

Default ports

The following are the default ports for services you can query through the status API endpoints. If you changed a service's port in your installation's configuration, you'll need to call that port instead.

Service	Port
Activity service	4433
Node classifier	4433
Code Manager, file sync client, and file sync storage	8140
Orchestrator, PCP broker, and PCP broker v2	8143
PuppetDB	8081
RBAC service	4433
Puppet Server	8140

Status API: services endpoint

The `/services` endpoints provide machine-consumable information about running services. They are intended for scripting and integration with other services.

GET /status/v1/services

Retrieves statuses for all Puppet Enterprise (PE) services on a specific port.

Request format

When [Forming status API requests](#) on page 418 to this endpoint, you must specify the port associated with the PE services you want to query. The default ports and their associated services are as follows:

Port	Service(s)
4433	<ul style="list-style-type: none"> • RBAC • Activity Service • Classifier
8140	<ul style="list-style-type: none"> • Code Manager • File sync client • File sync storage • Puppet Server • PCP broker (compilers) • PCP broker v2 (compilers)
8143	<ul style="list-style-type: none"> • Orchestrator • PCP broker (primary server) • PCP broker v2 (primary server)
8081	PuppetDB

Important: If you changed a service's port to something other than the default port, you might need to change the port number in your request.

This endpoint's content type is `application/json; charset=utf-8`, and you can append these parameters to the URL:

- `level`: How thorough of a check to run. Set to `critical`, `debug`, or `info`. The default is `info`.
- `timeout`: Specified in seconds. The default is 30.

For example, this request uses certificate authentication and fetches `info` status information for PE services running on port 4433:

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"

curl --cert "$cert" --cacert "$cacert" --key "$key" \
--header "Content-Type: application/json; charset=utf-8" \
--request GET "https://puppet.status.example:4433/status/v1/services?
level=info&timeout=60"
```

Response format

The server uses these 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 an invalid level parameter is set (not critical, debug, or info).

A successful response contains a JSON object listing details about the services. Responses use these 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	The level of detail shown. One of critical, debug, or info.
state	The current state of the service. One of 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, a request about services on port 4433 (which includes the Activity service, the Classifier, and RBAC) returns a response similar to the following:

```
{
  "activity-service": {
    "service_version": "2019.8.0.0",
    "service_status_version": 1,
    "detail_level": "info",
    "state": "running",
    "status": {
      "db_up": true,
      "db_pool": {
        "state": "ready"
      },
      "replication": {
        "mode": "none",
        "status": "none"
      }
    },
    "active_alerts": []
  },
  "classifier-service": {
    "service_version": "2019.8.0.0",
    "service_status_version": 1,
    "detail_level": "info",
    "state": "running",
    "status": {
      "db_up": true,
      "db_pool": {
        "state": "ready"
      },
      "rbac_up": true,
      "activity_up": true,
      "replication": {
        "mode": "none",
        "status": "none"
      }
    }
  }
},
```

```

        "active_alerts": [ ]
    },
    "rbac-service": {
        "service_version": "2019.8.0.0",
        "service_status_version": 1,
        "detail_level": "info",
        "state": "running",
        "status": {
            "db_up": true,
            "db_pool": {
                "state": "ready"
            },
            "activity_up": true,
            "replication": {
                "mode": "none",
                "status": "none"
            }
        },
        "active_alerts": []
    },
    "status-service": {
        "service_version": "1.1.0",
        "service_status_version": 1,
        "detail_level": "info",
        "state": "running",
        "status": {},
        "active_alerts": []
    }
}

```

GET /status/v1/services/<SERVICE NAME>

Retrieves the status of one Puppet Enterprise (PE) service.

Request format

When [Forming status API requests](#) on page 418 to this endpoint, your request must include a properly-formatted service name and the corresponding port. Service names and default ports are as follows:

Service	Port
activity-service	4433
broker-service	8143 (primary server) 8140 (compilers)
classifier-service	4433
code-manager-service	8140
orchestrator-service	8143
puppetdb-service	8081
rbac-service	4433
server (Puppet Server)	8140

Important: If you changed a service's port to something other than the default port, you might need to change the port number in your request.

This endpoint's content type is `application/json; charset=utf-8`, and you can specify these parameters in your request:

- `level`: How thorough of a check to run. Set to `critical`, `debug`, or `info`. The default is `info`.
- `timeout`: Specified in seconds. The default is 30.

For example, this request uses certificate authentication and fetches `info` status information for the RBAC service:

```
cert="$(puppet config print hostcert)"
cacert="$(puppet config print localcacert)"
key="$(puppet config print hostprivkey)"

curl --cert "$cert" --cacert "$cacert" --key "$key" \
--header "Content-Type: application/json; charset=utf-8" \
--request GET "https://puppet.example.com:4433/status/v1/services/rbac-
service?level=info&timeout=60"
```

Response format

The server uses these 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 an invalid `level` parameter is set (not `critical`, `debug`, or `info`).
- 404 if no service matching the supplied service name is found.

A successful response contains a JSON object listing details about the service using these keys:

Key	Definition
<code>service_version</code>	Package version of the JAR file containing a given service.
<code>service_status_version</code>	The version of the API used to report the status of the service.
<code>detail_level</code>	The level of detail shown. One of <code>critical</code> , <code>debug</code> , or <code>info</code> .
<code>state</code>	The current state of the service. One of <code>running</code> , <code>error</code> , or <code>unknown</code> .
<code>status</code>	An object with the service's status details. Usually only relevant for error and unknown states.
<code>active_alerts</code>	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, this response contains information about the RBAC service:

```
{
  "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": [ ]
    }
}
```

Status API: services plaintext endpoint

The status service plaintext endpoints are intended for load balancers that don't support JSON parsing or parameter setting. These endpoints return simple string bodies (either the service's state or a simple error message) and a relevant status code.

GET /status/v1/simple

Returns a cumulative status reflecting all services the status service knows about.

Request format

When [Forming status API requests](#) on page 418 to this endpoint, the content type for this endpoint is `text/plain; charset=utf-8`.

This endpoint supports no parameters. It uses the `critical` status level by default.

Response format

The server uses these response codes:

- 200 if, and only if, all services report a status of `running`.
- 503 if any service's status is `unknown` or `error`.

The response reflects a single, cumulative status of all services the endpoint is aware of. The endpoint uses this logic to determine which status to report:

- `running` if, and only if, all services report as `running`.
- `error` if any one service reports an `error`.
- `unknown` if any one service reports as `unknown` *and* no services report an `error`.

Therefore, while some services may be `running`, the status can still be `error` or `unknown` as long as any one service is not `running`. For example, if two services report as `running` and one service reports `unknown`, then the response is 503: `unknown`. If one service reports as `running`, one service reports `unknown`, and one service reports `error`, then the endpoint response is 503: `error`.

GET /status/v1/simple/<SERVICE NAME>

Returns a plaintext status for a specified service, such as the `rbac-service` or `classifier-service`.

Request format

When [Forming status API requests](#) on page 418 to this endpoint, the content type for this endpoint is `text/plain; charset=utf-8`. Your request must include a properly-formatted service name, such as:

- `activity-service`
- `classifier-service`
- `code-manager-service`
- `orchestrator-service`
- `puppetdb-service`
- `rbac-service`
- `server` (Puppet Server)

This endpoint supports no parameters. It uses the `critical` status level by default.

Response format

The server can return these response codes and messages:

- 200: running if the service is running.
- 503: error if the service reported an error.
- 503: unknown if the service reported as unknown.
- 404: not found: <SERVICE_NAME> if no service matching the supplied service name is found.

Status API: metrics endpoint

Puppet Server can track advanced metrics to give you additional insight into its performance and health.

Restriction: These API endpoints are a tech preview. These metrics are returned only when the request contains the `level=debug` parameter. The response structure might change in future versions.

There are three metrics endpoints:

- The `services/pe-jruby-metrics` endpoint returns JRuby metrics.
- The `services/pe-master` endpoint returns HTTP route metrics.
- The `services/pe-puppet-profiler` endpoint returns catalog compilation profiler metrics.

These metrics reflect data collected over the lifetime of the current Puppet Server process. Data resets when the service is restarted. All time-related metrics report in milliseconds unless otherwise noted.

GET /status/v1/services/pe-jruby-metrics

Returns JSON-formatted information about the JRuby pools from which Puppet Server fulfills agent requests.

Request format

The HTTPS metrics endpoints are available on port 8140 of the primary server. Your request must query port 8140 and include the `level=debug` parameter. For example:

```
uri="https://$(puppet config print server):8140/status/v1/services/pe-jruby-metrics?level=debug"
curl --insecure "$uri"
```

For information about `puppet config` commands and `curl` commands in Windows, go to [Using example commands](#) on page 28.

Response format

The server uses these response codes:

- 200 if, and only if, all services report a status of running.
- 503 if any service's status is unknown or error.

In addition to a response code, the metrics endpoints return machine-consumable (JSON-formatted) information about PE services. These JSON responses use the same keys returned by the [GET /status/v1/services](#) on page 419 and [GET /status/v1/services/<SERVICE NAME>](#) on page 422 endpoints. The metrics endpoints also return additional keys in the experimental section of the response.

Within the experimental section of the `pe-jruby-metrics` endpoint response, the metrics are divided into two subsections: `jruby-pool-lock-status` and `metrics`.

The `jruby-pool-lock-status` subsection contains these keys:

Key	Definition
current-state	The JRuby pool lock state: <ul style="list-style-type: none"> • :not-in-use (unlocked) • :requested (waiting for lock) • :acquired (locked)
last-change-time	The date and time of the last current-state update, formatted as an ISO 8601 combined date and time in UTC.

The metrics subsection contains these keys:

Key	Definition
average-borrow-time	The average time a JRuby instance spends handling requests. This is calculated by dividing the total duration in milliseconds of the borrowed-instances value by the borrow-count value.
average-free-jrubies	The average number of JRuby instances not used over the Puppet Server process's lifetime.
average-lock-held-time	The average time the JRuby pool held a lock, starting when the value of jruby-pool-lock-status/current-state changed to :acquired. This time mostly represents the file sync service syncing code into the live codedir, and it is calculated by dividing the total length of time Puppet Server held the lock by the num-pool-locks value.
average-lock-wait-time	The average time Puppet Server spent waiting to lock the JRuby pool, starting when the value of jruby-pool-lock-status/current-state changed to :requested. This time mostly represents how long Puppet Server takes to fulfill agent requests, and it is calculated by dividing the total length of time Puppet Server waits for locks by the num-pool-locks value.
average-requested-jrubies	The average number of requests waiting on an available JRuby instance over the Puppet Server process's lifetime.
average-wait-time	The average time Puppet Server spends waiting to reserve an instance from the JRuby pool. It is calculated by dividing the total duration, in milliseconds, of requested-instances by the requested-count value.
borrow-count	The total number of JRuby instances that have been used.
borrow-retry-count	The total number of times a borrow attempt failed and was retried, such as when the JRuby pool is flushed while a borrow attempt is pending.

Key	Definition
borrow-timeout-count	The number of requests that were not served because they timed out while waiting for a JRuby instance.
borrowed-instances	<p>A list of JRuby instances currently in use, with each reporting:</p> <ul style="list-style-type: none"> duration-millis: The length of time that the instance has been running. reason/request: A hash of details about the request being served. <ul style="list-style-type: none"> request-method: The HTTP request method, such as POST, GET, PUT, or DELETE. route-id: The route being served. For routing metrics, see the HTTP metrics endpoint. uri: The request's full URI. time: The time (in milliseconds, since the Unix epoch) when the JRuby instance was borrowed.
num-free-jrubies	The number of JRuby instances in the pool that are ready to be used.
num-jrubies	The total number of JRuby instances.
num-pool-locks	The total number of times the JRuby pools have been locked.
requested-count	The number of JRuby instances borrowed, waiting, or that have timed out.
requested-instances	<p>A list of requests waiting to be served, with each reporting:</p> <ul style="list-style-type: none"> duration-millis: The length of time the request has waited. reason/request: A hash of details about the waiting request. <ul style="list-style-type: none"> request-method: The HTTP request method, such as POST, GET, PUT, or DELETE. route-id: The route being served. For routing metrics, see the HTTP metrics endpoint. uri: The request's full URI. time: 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.

Here is an example response for the `pe-jruby-metrics` endpoint:

```

"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/
hostname.example.com"
                }
              }
            },
            "time": 1448478371406
          }
        ],
        "num-free-jrubies": 0,
        "num-jrubies": 1,
        "num-pool-locks": 2849,
        "requested-count": 640,
        "requested-instances": [
          {
            "duration-millis": 3663,
            "reason": {
              "request": {
                "request-method": "put",
                "route-id": "puppet-v3-report-/*/",
                "uri": "/puppet/v3/report/
hostname.example.com"
                }
              }
            },
            "time": 1448478371715
          }
        ],
        "return-count": 638
      }
    }
}

```

GET /status/v1/services/pe-master

Returns JSON-formatted information about the routes that agents use to connect to the primary server.

Request format

The HTTPS metrics endpoints are available on port 8140 of the primary server. Your request must query port 8140 and include the `level=debug` parameter. For example:

```
uri="https://$(puppet config print server):8140/status/v1/services/pe-
master?level=debug"

curl --insecure "$uri"
```

For information about `puppet config` commands and `curl` commands in Windows, go to [Using example commands](#) on page 28.

Response format

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`.

In addition to a response code, the metrics endpoints return machine-consumable (JSON-formatted) information about PE services. These JSON responses use the same keys returned by the [GET /status/v1/services](#) on page 419 and [GET /status/v1/services/<SERVICE NAME>](#) on page 422 endpoints. The metrics endpoints also return additional keys in the `experimental` section of the response.

Within the `experimental` section of the `pe-master` endpoint response, there is one subsection, `http-metrics`, containing these keys:

Key	Definition
<code>aggregate</code>	The total time Puppet Server spent processing requests for this route.
<code>count</code>	The total number of requests Puppet Server processed for this route.
<code>mean</code>	The average time Puppet Server spent on each request for this route, calculated by dividing the <code>aggregate</code> value by the <code>count</code> value.
<code>route-id</code>	The route being served. The request returns a route with the special <code>route-id</code> of <code>total</code> , which represents the aggregate data for all requests along all routes. If not <code>total</code> , values use the <code>puppet-v3</code> prefix, such as <code>puppet-v3-report/*/</code> .

GET /status/v1/services/pe-puppet-profiler

Returns JSON-formatted 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.

Request format

The HTTPS metrics endpoints are available on port 8140 of the primary server. Your request must query port 8140 and include the `level=debug` parameter. For example:

```
uri="https://$(puppet config print server):8140/status/v1/services/pe-
puppet-profiler?level=debug"
```

```
curl --insecure "$uri"
```

For information about `puppet config` commands and `curl` commands in Windows, go to [Using example commands](#) on page 28.

Response format

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`.

In addition to a response code, the metrics endpoints return machine-consumable (JSON-formatted) information about PE services. These JSON responses use the same keys returned by the [GET /status/v1/services](#) on page 419 and [GET /status/v1/services/<SERVICE NAME>](#) on page 422 endpoints. The metrics endpoints also return additional keys in the `experimental` section of the response.

The Puppet Server profiler is enabled by default, but if it is disabled, there are no metrics available for the `pe-puppet-profiler` endpoint to return. In this case, the endpoint returns the same keys as [GET /status/v1/services](#) on page 419 and an empty `status` key.

If the profiler is enabled, within the `experimental` section of the `pe-puppet-profiler` endpoint response, the metrics are divided into two subsections:

- `function-metrics`: Contains statistics about functions evaluated by Puppet Server when compiling catalogs.
- `resource-metrics`: Contains statistics about resources declared in manifests compiled by Puppet Server.

You'll find these keys in the two subsections:

Key	Definition
<code>function or resource</code>	Each function measured in the <code>function-metrics</code> section has a <code>function</code> key containing the function's name.
<code>aggregate</code>	Each resource measured in the <code>resource-metrics</code> section has a <code>resource</code> key containing the resource's name.
<code>count</code>	The total time spent handling the function call or resource during catalog compilation.
<code>mean</code>	The number of times during catalog compilation that Puppet Server has called the function or instantiated the resource.

Here is an example response for the `pe-puppet-profiler` endpoint:

```
"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]"
        },
        {...},
    ]
}
}
```

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 432

You can add nodes you want to manage with Puppet Enterprise (PE) and remove nodes you no longer need.

- [Adding and removing agentless nodes](#) on page 433

Using the inventory service, you can manage nodes and devices (such as network switches and firewalls) without installing the Puppet agent on them. Node and device information is stored securely in your Puppet Enterprise (PE) inventory.

- [How nodes are counted](#) on page 437

Your *node count* is the number of nodes in your inventory. Your Puppet Enterprise (PE) 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 438

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 trigger a Puppet run.

- [Grouping and classifying nodes](#) on page 440

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

- [Making changes to node groups](#) on page 449

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 451

The environment-based testing workflow lets you test new code before pushing it to production.

- [Preconfigured node groups](#) on page 455

Puppet Enterprise includes preconfigured node groups that are used to manage your configuration.

- [Managing Windows nodes](#) on page 459

You can use Puppet Enterprise (PE) to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks with modules from the Forge.

- [Designing system configs \(roles and profiles\)](#) on page 486

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 513

These are the endpoints for the node classifier v1 API.

- [Node classifier API v2](#) on page 563

These are the endpoints for the node classifier v2 API.

- [Node inventory API v1](#) on page 566

These are the endpoints for the node inventory v1 API.

Adding and removing agent nodes

You can add nodes you want to manage with Puppet Enterprise (PE) and remove nodes you no longer need.

To add nodes:

1. [Install agents](#) on the nodes.
2. Accept the CSRs, as explained in [Managing certificate signing requests](#) on page 164.

Related information

[Upgrading agents](#) on page 196

Upgrade your agents as new versions of Puppet Enterprise (PE) become available. The `puppet_agent` module helps automate upgrades, and provides the safest upgrade. Alternatively, you can use a script to upgrade individual nodes.

[Set a proxy for agent traffic](#) on page 229

General proxy settings in an agent node's `puppet.conf` file are used to manage HTTP connections directly initiated by the agent node.

Remove agent nodes

Purging a node removes it from your inventory so it is no longer managed by Puppet Enterprise (PE) and allows you to use the node's license on 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.

Restriction: Removing (purging) nodes doesn't [Uninstall agents](#) on page 177 from the nodes.

1. On the agent node, run this command to stop the agent service: `service puppet stop`
2. On the primary server, run this command to 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 agent -t` on your compilers to distribute the updated CRL to them.
4. Optional: If the node you removed was pinned to any node groups, you must manually unpin it from the individual node groups (or from all node groups) using the `unpin-from-all` command.

Related information

[Uninstall infrastructure nodes](#) on page 176

The `puppet-enterprise-uninstaller` script is installed on the primary server. You must run the uninstaller script on each infrastructure node you want to uninstall.

[Uninstall agents](#) on page 177

You can remove the `puppet-agent` package from nodes that you no longer want Puppet Enterprise (PE) to manage.

[Uninstaller options](#) on page 178

You can use these command line options to change the uninstaller's behavior.

[POST /v1/commands/unpin-from-all](#) on page 543

Unpin one or more specific nodes from all node groups they're pinned to. Unpinning has no effect on nodes that are assigned to node groups via dynamic rules.

Adding and removing agentless nodes

Using the inventory service, you can manage nodes and devices (such as network switches and firewalls) without installing the Puppet agent on them. Node and device information is stored securely in your Puppet Enterprise (PE) inventory.

- *Agentless nodes* are nodes that don't have a Puppet agent installed on them. They can do things like run tasks and plans, but they do not help maintain your infrastructure's desired state in the way [agent nodes](#) do.
- *Devices* or *agentless device* are devices, such as network switches or firewalls, that can't have a Puppet agent installed on them. Connecting devices lets you manage these network device and run Puppet and task jobs on them.

The inventory service uses SSH or WinRM remote connections to connect to agentless nodes. To connect to agentless devices, the inventory service uses transport definitions from device transport modules you've installed.

After you add agentless node or device credentials to the inventory, authorized users can run tasks on the agentless nodes and devices without re-entering the credentials. On the **Tasks** page (in the console), the agentless nodes and devices are listed together with the nodes and devices that have an agent installed.

Add agentless nodes to the inventory

Use SSH or WinRM remote connections to add agentless nodes to your Puppet Enterprise (PE) inventory so you can run tasks on them. Agentless nodes are nodes that can't (or don't) have a Puppet agent installed 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 your user account has this permission: **Nodes: Add and delete connection information from inventory service**

1. In the PE console, click **Nodes > Add nodes**.
2. Click **Connect over SSH or WinRM**.
3. Select a transport method.
 - **SSH** for *nix and macOS targets
 - **WinRM** for Windows targets
4. Enter target host names and the credentials required to access them. If you use an SSH key, include the begin and end tags.
5. Optional: Select additional [Transport configuration options](#) on page 434. For example, to customize the connection port number, select **Target Port** from the **Target options** drop-down list, enter the desired port number, and click **Add**.
6. Click **Add nodes**.

After adding agentless nodes to your PE inventory, they are added to PuppetDB, and you can view them on the **Nodes** page (in the console). Any nodes in your inventory can be added to the inventory node list when you set up a job to run tasks. To review a node's connection settings or remove an agentless node from the inventory, go to the **Connections** tab on the **Node details** page.

Transport configuration options

Descriptions of the target options for SSH and WinRM transports.

Option	Transport method	Definition
Target port	SSH and WinRM	The connection port. For SSH, the default is 22. For WinRM, the default is 5986, unless <code>ssl: false</code> , then the default is 5985.
Connection time-out in seconds	SSH and WinRM	The length of time you want Puppet Enterprise (PE) to wait for a response when attempting to establish a connection.
Temporary directory	SSH and WinRM	The directory to use when uploading temporary files to the target node.

Option	Transport method	Definition
Run as another user	SSH	After login, this is the user profile to use for running commands.
Sudo password	SSH	The password to use when switching user profiles via <code>run-as</code> .
Process request as tty	SSH	Use this if you need to enable text terminal allocation.
Acceptable file extension	WinRM	A list of allowed file extensions for scripts or tasks. Scripts with the specified file extensions rely on the target node's file type associations to run. For example, if Python is installed on the target node, a <code>.py</code> script from PE uses <code>python.exe</code> to run (unless the file type association was changed on the target node). Tip: 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

By adding devices to your Puppet Enterprise (PE) inventory, you can manage network devices, such as switches and firewalls, and run Puppet and task jobs on them, just like the agentless nodes in your infrastructure.

Before you begin

Depending on the device you want to connect, you must install the appropriate device transport module in your PE production environment before you can add the device to your inventory. You can find device modules on the Puppet Forge, such as the `panos` and `cisco_ios` modules.

Make sure your user account has this permission: **Nodes: Add and delete connection information from inventory service**

Important: Managing more than 100 devices might cause performance issues on the primary server.

1. In the PE console, click **Nodes > Add nodes**.
2. Click **Connect network devices**.

3. Select the device type from the list of device transport modules that you have installed in your production environment.

If no device types are available, or the relevant device type is missing, check that:

- You have installed the appropriate module for the device you want to manage.
- The module is installed correctly.
- The module is installed in your production environment.
- Your Puppet code has been deployed. If you're using Code Manager or r10k for [Managing and deploying Puppet code](#) on page 762, you might need to trigger a code deployment.

For information about modules and installing modules, refer to the [Modules overview](#) and [Installing and managing modules from the command line](#) in the Puppet documentation.

4. Enter the device certname and other connection details, as specified in the transport module's README on the [Forge](#). Mandatory fields are marked with an asterisk.

5. Click **Add node**.

After adding devices to your PE inventory, they are added to PuppetDB, and you can view them on the [Nodes](#) page (in the console). Any devices in your inventory can be added to the inventory node list when you set up a job to run tasks. To review a device's connection settings or remove a device from the inventory, go to the [Connections](#) tab on the device's [Node details](#) page.

Related information

[Managing modules with a Puppetfile](#) on page 768

Almost all Puppet manifests are kept in *modules*, which are collections of Puppet code and data that have a specific directory structure. With Puppet Enterprise (PE) code management, you only use the Puppetfile to install and manage modules.

Remove devices and agentless nodes from the inventory

You can remove a device or agentless node from the Puppet Enterprise (PE) inventory by going to the [Connections](#) tab on the [Node details](#) page. This can also be referred to as disconnecting the node or device.

Before you begin

Make sure your user account has this permission: **Nodes: Add and delete connection information from inventory service**

1. In the Puppet Enterprise (PE) console, click **Status** or **Nodes**, find the node or device you want to remove, and click its name to open the [Node details](#) page.
2. Switch to the [Connections](#) tab.
3. Click **Remove connection**. This link's name depends on the connection type, such as **Remove SSH Connection**, **Remove WinRM connection**, and so on.
4. Confirm that you want to remove the connection.

When you remove a node or device from the inventory, PuppetDB marks the node or device as expired after the standard node time-to-live period (`node-ttl`). Then PuppetDB purges the node or device when it reaches the node-purge time-to-live limit (`node-purge-ttl`). Once purged, the node or device no longer appears in the PE console, and the node's license is available to reassign to another node.

Tip: For more information about the `node-ttl` and `node-purge-ttl` settings, refer to the PuppetDB [\[database\] settings](#) in the Puppet documentation.

Related information

[Node inventory API v1](#) on page 566

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 Puppet Enterprise (PE) 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: On this page, the term *node* includes agent nodes, agentless nodes, primary servers, compilers, nodes running in noop mode, and purged nodes that had prior activity within the relevant 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 the nodes 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 tracked 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 exceed your license's node count limit, you reach the *bursting limit*. The bursting limit is a margin that allows you to temporarily exceed the number of nodes allowed by your license, and enter a new threshold, without extra cost. You are allowed to reach the bursting limit on four consecutive or non-consecutive days per calendar month. If you reach the bursting limit on five or more days, you must either purge nodes or buy a license for more nodes.

The amount of time your inventory exceeds the bursting limit does not matter for it to be counted as one day, whether it is one hour or several hours. For example, assuming your license allows 1000 nodes:

- If you use 1200 nodes for two hours on one day, you have three days left on your monthly bursting limit.
- If you use 1900 nodes for 23 hours on one day, you have three days left on your monthly bursting limit.
- If you use 1500 nodes for one hour per day for four days within a single calendar month, you've exhausted your monthly bursting limit. You must either purge nodes (until the next calendar month starts) or permanently raise your node limit (by contacting your Puppet representative and buying a license for more nodes).

When nodes are counted

PE tracks daily node counts from 12:00 midnight UTC to 12:00 midnight UTC. Therefore, if your node count exceeds the bursting limit at 23:00 UTC, and remains in excess of the bursting limit until 1:00 UTC, this would count as two of the four days allowed for your monthly bursting limit.

The same time is used to calculate the duration of each calendar month. For example, the month of September includes activity from 12:00 midnight UTC on 01 September until 12:00 midnight UTC on 01 October. At 12:00 midnight UTC on the first day of the next month, the bursting limit allowance resets to four days. For example, if your node count exceeds the bursting limit at 22:00 UTC on 30 September, and remains in excess of the bursting limit until 1:00 UTC on 01 October, this counts as one day towards the September bursting limit allowance and one day towards the October bursting limit allowance.

Viewing your node count

To view your daily node count in the PE console, go 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 query daily node usage information on the command line, use the orchestrator API [Usage endpoints](#) on page 753.

Removing nodes

If you have unused nodes cluttering your inventory, and you are concerned about reaching your bursting limit, you can [Remove agent nodes](#) on page 433 and [Remove devices and agentless nodes from the inventory](#) on page 436.

Related information

[Purchasing and installing a license key](#) on page 144

A Puppet Enterprise (PE) license includes access to Security Compliance Management (formerly Comply) and Continuous Delivery, both of which can be installed and used after you have installed PE. To unlock additional premium features including Security Compliance Enforcement (formerly CEM) and advanced Impact Analysis capabilities in Continuous Delivery, you can [contact our sales team](#).

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 trigger a Puppet run.

In a Puppet run, the primary server and agent nodes perform these actions:

1. Each agent node sends facts to the primary server and requests a catalog.
2. The primary server compiles and returns each agent's catalog.
3. Each agent applies the catalog by checking each resource the catalog describes. If the agent finds any resources that are not in the desired state, the agent makes the necessary changes to bring the resource into the desired state.

Note: Puppet run behavior differs slightly if static catalogs are enabled.

Related information

[Static catalogs](#) on page 245

A catalog is a document that describes the desired state for each resource that Puppet manages on a node. Puppet Enterprise (PE) primary servers typically compile catalogs from manifests of Puppet code. A static catalog is a specific type of Puppet catalog that includes metadata specifying 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 you can use to deploy configuration changes when and how you desire.

You can use the orchestrator to enforce change on a selection of nodes identified by their certnames, a PQL query, or a node group.

You can use the orchestrator from the console, command line, or through the orchestrator API endpoints. The orchestrator API is useful if you're putting together your own tools for running Puppet or if you want to enable CI workflows across your infrastructure.

Related information

[Run Puppet on demand from the console](#) on page 604

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 the targeted nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

[Run Puppet on demand from the CLI](#) on page 611

Use the `puppet job run` command to start an on-demand Puppet run to enforce changes on your agent nodes.

[Orchestrator API v1](#) on page 678

You can use the orchestrator API to run jobs and plans on demand; schedule tasks and plans; get information about jobs, plans, and events; track node usage; and more.

Running Puppet with SSH

To use trigger a Puppet run with SSH from an agent node, SSH into the target node and run `puppet agent --test` or `puppet agent -t`.

Running Puppet from the console

In the console, you can run Puppet from an agent node's **Node details** page.

Restriction: 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 or WinRM (an agentless node), or if it is a device.

1. In the console, go to **Nodes** and click the name of the node you want to run Puppet on.
2. On the **Node details** page, click **Run Puppet**. You can configure these run options, if desired:
 - **No-op**: The Puppet run simulates changes without actually enforcing the new catalog. Nodes with `noop = true` in their `puppet.conf` files always run in no-op mode.
 - **Debug**: Prints all messages generated during the run that are 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 Puppet run completes, the console displays the node's run status.

Related information

[Node run statuses](#) on page 382

The **Status** page displays each node's run status for the most recent Puppet run. Possible statuses depend on the Puppet run mode.

Activity logging when running Puppet from the console

When you initiate a Puppet run from the console, the Activity service logs the run activity.

You can view activity for a single node by opening the node's **Node details** page and switching to the **Activity** tab.

Alternatively, you can use the Activity Service API to retrieve activity information.

Related information

[Activity service API](#) on page 368

The activity service records changes to role-based access control (RBAC) entities, such as users, directory groups, and user roles. Use the activity service API to query event data.

Troubleshooting Puppet run failures

Puppet Enterprise (PE) creates a **View Report** link for most failed runs, which you can use to access the run's events and logs. You might encounter these errors when a Puppet run fails.

Changes could not be applied

Usually caused by conflicting classes. Check the run log to get information.

This error can also occur when running in no-op mode.

Run already in progress

Occurs when you try to trigger a Puppet run on a node, but there is already a Puppet run in progress. This could be a scheduled run or a run started by another user.

Run request times out

Occurs if you attempt to start a Puppet run but the agent isn't available.

Report request times out

Occurs when the run report is not successfully stored in PuppetDB after the run completes.

Invalid response, such as a 500 error

Some part of the request is invalid. If you used the command line or the orchestrator API to start the Puppet run, check the formatting of your command or request. If you're using the console, or your command or request is well-formed, your Puppet code might be have incorrect formatting.

In the console, the Run button is disabled and a run is not allowed.

You have permission to run Puppet on the node, but the agent is not responding.

Grouping and classifying nodes

Configure nodes by assigning classes, parameters, and variables to them. This is called *classification*.

To classify nodes, you must:

1. [Create node groups](#) on page 441 to contain nodes and preferences (classes, parameters, and variables) you want to apply to nodes in the group. Make sure you understand the [Best practices for classifying node groups](#) on page 441 and the difference between [Environment versus classification node groups](#) on page 441.
2. [Add nodes to groups](#).
3. [Declare classes](#) on page 445 and [Define data used by node groups](#) on page 446. You might need to [Enable data editing in the console](#) on page 235.

Nodes can belong to multiple node groups, and they [inherit](#) classes, class parameters, and variables from all node groups they belong to.

After classifying nodes, you can [View nodes in a node group](#) on page 449 and [Make changes to node groups](#).

How node group inheritance works

Node groups are organized in a parent-child-grandchild hierarchy. When added to a group, nodes inherit classes, parameters, variables, and rules from their immediate node group and the group's ancestors.

- [Classes](#): If a class is declared in an ancestor node group, the class is inherently declared in all node groups descending from that group.
- [Class parameters and variables](#): Descendant node groups inherit class parameters and variables from ancestors unless a different value is set in a descendant node group.



CAUTION: Because nodes can belong to multiple groups in separate hierarchies, it's possible for two node groups to contribute conflicting variable or class parameter values. Conflicting values cause Puppet runs on agent nodes to fail.

- [Dynamic node group rules](#): To belong to a group, a node must match all rules in the immediate group and the rules in the ancestor groups. Use rule inheritance to refine group membership: Create broad rules for parent groups (such as an OS family) and more specific rules for the child groups that refine the parent group rules (such as specific platforms or versions). This way you can take a large, generic set of nodes and filter them into specific child groups.

Tip: In the console, go to [Node groups](#) to see how node groups are related. The [Node groups](#) page shows a hierarchical view of your 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 551

Retrieve a list of node groups descending from a specific node group.

Best practices for classifying node groups

To organize node groups, start with the high-level functional groups that reflect the business requirements of your organization, and work down to smaller segments within those groups.

For example, if a large portion of your infrastructure consists of web servers, create a node group called `web servers` and add any classes that need to be applied to all web servers.

Next, identify subsets of web servers that have common characteristics but differ from other subsets. For example, you might have production web servers and development web servers. So, create a `dev web` child node group under the `web servers` node group. Nodes that match the `dev web` node group get all of the classes in the parent node group in addition to the classes assigned to the `dev web` node group.

Environment versus classification node groups

Environment node groups assign environments to nodes, such as `test`, `development`, or `production`.

Important: A node can belong to only one environment node group. If a node is added to more than one environment group, classification errors occur. See this [classification conflict article](#) for more information.

Each environment node group:

- Must correspond to a Git branch in a control repo you want to use for targeted code deployments. The Git branch and environment group must have the same name.
- Must be a child of the **All Environments** node group (or whichever is the highest-level environment node group in your installation). Furthermore, your environment node groups, themselves, **must not** have any child groups, except one-time run exception subgroups used for canary testing.
- Must not include classes or configuration data.

Classification node groups assign classification data to nodes, including classes, parameters, and variables. A node can belong in more than one classification group.

Each classification node group:

- Must be a child of **All Nodes** or another classification group.
- Must not be specified as an environment group in the group metadata.

Related information

[Environment-based testing](#) on page 451

The environment-based testing workflow lets you test new code before pushing it to production.

Create node groups

Use the console to create node groups to assign either an environment or classification.

Create environment node groups

Create custom environment node groups so you can target Puppet code deployments.

Before you begin

Each environment node group must correspond to a Git branch in a control repo you want to use for targeted code deployments. The Git branch and environment group must have the same name. Make sure you know the names of the corresponding Git branches for your environment node groups.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the new node group:

- **Parent name:** Select the top-level environment node group in your hierarchy, such as **All environments** or **Production environment** (depending on your installation's configuration).

Important: Each environment mode group must be a child of the highest-level environment node group in your installation.

- **Group name:** Enter a name corresponding to the Git branch in your control repo that you want to use for targeted code deployments for this environment.

Important: The Git branch and environment group must have the same name.

- **Environment:** Select the environment that you want to assign to the nodes in this node group.
- **Environment group:** Select this option.

3. Click Add.

Add nodes to your environment node group to control which environment each node belongs to. Each node can belong to only one environment node group.

Related information

[Environment-based testing](#) on page 451

The environment-based testing workflow lets you test new code before pushing it to production.

Create classification node groups

Classification node groups assign classification data to nodes.

1. In the console, click **Node groups**, and click **Add group**.

2. Specify options for the new node group:

- **Parent name:** Select the classification node group that you want to be the parent of your new classification node group. Classification node groups inherit classes, parameters, and variables from their parent node group. The default parent node group is the **All Nodes** node group.
- **Group name:** Enter a name that describes the classification node group's role. For example, **Web Servers**.
- **Environment:** Specify an environment to limit the classes and parameters available for selection in this node group.

Specifying the **Environment** in a classification node group only filters the available classes and parameters. It does not assign an environment to the nodes in the group (as is the case in [environment node groups](#)).

- **Environment group:** Do not select this option.

3. Click Add.

Dynamically or statically add nodes to the classification node group.

Add nodes to a node group

There are two ways to add nodes to a node group.

You can [Statically add individual nodes to a node group](#) or use fact-based rules to [Dynamically add nodes to a node group](#) on page 443. With dynamic node group rules, Puppet Enterprise (PE) automatically adds and removes nodes from your groups based on the rules you set, whereas you must manually add or remove static (pinned) nodes.

If you expect a node to belong to multiple node groups, make sure you understand [How node group inheritance works](#) on page 440. If nodes inherit conflicting values from different groups, then Puppet runs on agent nodes fail.

Statically add nodes to a node group

You can pin individual nodes to node groups.

If you need to add a lot of nodes to a group, it is more useful to [Dynamically add nodes to a node group](#) on page 443 with logical rules that automatically add and remove nodes from your node groups. Pinning individual nodes to groups is only recommended if the node group only has a few nodes or if you need to add some specific nodes that weren't captured by the group's dynamic rules.

Pinning a node is the same as creating a rule for `certname = <EXACT_NODE_CERTNAME>`, and Puppet Enterprise (PE) processes this rule along with the group's other fact-based rules (if there are any).

Important: A pinned node remains in the node group until you manually remove it.

1. In the console, click **Node groups** and select the node group that you want to pin the node to.
2. On the **Rules** tab, enter the node's certname in the **Certname** field.
3. Click **Pin node** and commit changes.

Pinned nodes are listed under the **Certname** field.

Related information

[How node group inheritance works](#) on page 440

Node groups are organized in a parent-child-grandchild hierarchy. When added to a group, nodes inherit classes, parameters, variables, and rules from their immediate node group and the group's ancestors.

[Remove nodes from a node group](#) on page 449

To remove nodes from a node group, you must either unpin the node or delete (or change) the dynamic rule that added the node.

Dynamically add nodes to a node group

Rules are the most powerful and scalable way to include nodes in a node group. Rules use facts to identify nodes to include in a group.

Rules are based on facts, such as operating system, BIOS version, hardware model, UUID, or time zone. In addition to most [core facts](#), you can also [use structured and trusted facts](#) for node group rules.

As long as a node matches the node group's rules, the node is included in the group and classified with the node group's classification data (classes, parameters, and variables). If the node changes and no longer matches the node group's rules, the node is no longer considered part of the group, and the node group's classification data no longer applies to the node.

1. In the console, click **Node Groups** and select the node group you want to add nodes to.
 2. Think about which nodes you want to add to this group and the characteristics of those nodes. Rules are based on [Facter facts](#), so you must determine which facts describe the nodes you want (or don't want) in this group. Then, you can create logical rules based on those facts.
- Rules can be inclusive or exclusive, and you can apply multiple rules to each node group. When you have multiple rules, you can require nodes to match all rules or only match one of the rules.
3. On the **Rules** tab, create a fact-based rule, and click **Add Rule**.

For example, this rule specifies that nodes in the group must have a Red Hat OS:

- **Fact:** `osfamily`
- **Operator:** `=`
- **Value:** `RedHat`

[Writing node group rules](#) on page 444 explains the various **Operator** options and how to select core, structured, and trusted facts.

4. If needed, add more rules, and select how to apply the rules: select to Or, select to
- **Nodes must match all rules:** Combine rules for more granular node selection.
- **Nodes may match any rule:** Add a variety of rules to select nodes with different characteristics.

Tip: If you have a few individual nodes that you aren't able to capture with rules, you can [Statically add nodes to a node group](#) on page 442 in addition to your dynamic rules.

5. Commit changes.

Related information

[How node group inheritance works](#) on page 440

Node groups are organized in a parent-child-grandchild hierarchy. When added to a group, nodes inherit classes, parameters, variables, and rules from their immediate node group and the group's ancestors.

[Remove nodes from a node group](#) on page 449

To remove nodes from a node group, you must either unpin the node or delete (or change) the dynamic rule that added the node.

Writing node group rules

To create a dynamic node group rule, you must specify a **Fact**, **Operator**, and **Value**.

These three fields are required to [Dynamically add nodes to a node group](#) on page 443.

The **Fact** field specifies the fact you want to the rule to use. You can select from the dropdown list of known facts, enter part of a string to filter the list, or [use structured or trusted facts](#). For structured and trusted facts, select the initial value from the dropdown list, then type the rest of the fact:

- To descend into a hash, use periods (.) to designate path segments, such as `os.release.major` or `trusted.certname`.
- To specify an item in an array, put square brackets around the item's numerical index, such as `processors.models[0]` or `mountpoints./options[0]`.
- To identify path segments that have periods, dashes, spaces, or UTF characters, put single or double quotes around the segment, such as `trusted.extensions."1.3.6.1.4.1.34380.1.2.1"`.
- To use `trusted.extensions` short names, append the short name after a second period, such as `trusted.extensions.pp_role`.

Important: When you enter structured and trusted facts, PE doesn't provide suggestions as you type (except for the top-level name key), nor does it verify that the facts exist. After you add a rule that uses a structured or trusted fact, check the number of **Node matches** to verify that the fact is generating matches and was entered correctly.

The **Operator** field describes the relationship between the **Fact** and the **Value**. It defines how the rule uses the fact (inclusively, exclusively, exactly, minimum, maximum, and so on). Operators include:

- `=`: Is
- `!=`: Is not
- `~`: Matches regex
- `!~`: Does not match regex
- `>`: Greater than
- `>=`: Greater than or equal to
- `<`: Less than
- `<=`: Less than or equal to

The `>`, `>=`, `<`, and `<=` operators require facts that have numeric values, such as dates, times, or version numbers.

The `~` and `!~` operators require a regular expression (regex) **Value** and are useful for matching highly-specific node facts (for example, you want to find nodes with UUIDs starting with `9999*`)

The **Value** field describes the relevant **Fact** value. Depending on the **Fact** and **Operator**, the **Value** can be a string, number, or regular expression.

Together these fields create a logical rule that PE uses to find matching nodes. If you have multiple dynamic rules for one node group, you can choose whether **Nodes must match all rules** or if **Nodes may match any rule** to be added to the group.

Using structured and trusted facts for node group rules

Structured facts group related facts, and trusted facts are a type of structured fact.

[Structured facts](#) group related facts in a hash or array. For example, the `os` structured fact contains multiple individual facts about the operating system, such as architecture, family, and release. In the Puppet Enterprise (PE) console, when you view a node's facts, structured facts are surrounded by curly braces.

Trusted facts are a type of structured fact where the facts are immutable and extracted from a node's certificate. Because these facts 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 [dynamic node group rules](#).

Restriction: If you use trusted facts to specify certificate extensions, in order for this fact to function properly in a node group rule, you must use short names for [Puppet-specific registered IDs](#) and numeric IDs for [private extensions](#). Private extensions require numeric IDs whether or not you specify a short name in the `custom_trusted_oid_mapping.yaml` file.

Declare classes

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

Before you begin

The class that you want to apply must exist in an installed module. You can download modules from the Puppet Forge or create your own module. For information about modules and installing modules, refer to the [Modules overview](#) and [Installing and managing modules from the command line](#) in the Puppet documentation.

1. In the Puppet Enterprise (PE) console, click **Node groups** and select the node group that you want to add the class to.
2. On the **Classes** tab, select the class to add.

Tip: The **Add new class** field suggests classes that your PE primary server knows about and that are available in the environment defined in the node group's settings.

If classes are missing, check that:

- You have correctly installed the module containing the class you want to assign.
- The module is installed in the environment defined in the node group's settings. For information about the **Environment** setting for classification node groups, refer to [Create classification node groups](#) on page 442.
- Your Puppet code has been deployed since installing the module or making changes to node groups. If you're using Code Manager or r10k for [Managing and deploying Puppet code](#) on page 762, you might need to trigger a code deployment.

3. Click **Add class** and then commit changes.

Tip: 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

In new Puppet Enterprise (PE) installations, you can, by default, edit configuration data in the console. If you upgraded from an earlier PE version where you hadn't already enabled editing of configuration data, you must use Hiera to manually enable **Classifier Configuration Data**.

1. On your primary server, open the `hiera.yaml` file located at: `/etc/puppetlabs/puppet/hiera.yaml`.
2. Add the following to the `hiera.yaml` file:

```
hierarchy:
  - name: "Classifier Configuration Data"
    data_hash: classifier_data
```

Place additional hierarchy entries, such as `hiera-yaml` or `hiera-eyaml` under the same `hierarchy` key, below the `Classifier Configuration Data` entry.

3. To allow users to edit the configuration data in the console, add the **Set environment** and **Edit configuration data** permissions to any user groups that need to set environment parameters or modify class parameters.
4. If your environment is configured for disaster recovery or has compilers, update `hiera.yaml` on your replica and compilers, respectively.

Define data used by node groups

The console offers multiple ways to specify data used in your manifests.

- [Set configuration data](#) on page 446: Specify values through automatic parameter lookup.
- [Set parameters](#) on page 212: Specify resource-style values used by a [declared class](#).
- [Set variables](#) on page 447: Specify values to make available in Puppet code as top-scope variables.

Important: You can structure parameters and variables as JSON, but, if they can't be parsed as JSON, they're treated as strings.

Related information

[Tips for specifying parameter and variable values](#) on page 447

Parameters and variables can be structured as JSON, but, if they can't be parsed as JSON, they're treated as strings.

Set configuration data

Configuration data set in the PE console is used for automatic parameter lookup in 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 permission to define or edit 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 enable data editing in the console for both your primary server and replica.

1. In the console, click **Node groups** and select the node group that you want to add configuration data to.
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: If necessary, change the parameter's default **Value**.

Related information

[Enable data editing in the console](#) on page 235

In new Puppet Enterprise (PE) installations, you can, by default, edit configuration data in the console. If you upgraded from an earlier PE version where you hadn't already enabled editing of configuration data, you must use Hiera to manually enable [Classifier Configuration Data](#).

Set parameters

Parameters are declared resource-style, which means you can use them to override other data; however, this override capability can introduce class conflicts and declaration errors that cause Puppet runs to fail.

Important: You can structure parameters as JSON, but, if they can't be parsed as JSON, they're treated as strings.

1. In the console, click **Node groups** and select the node group you want to add a parameter to.

- On the **Classes** tab, select the class you want to modify, and select the **Parameter** you want to add.

The **Parameter** list shows all parameters available for the selected class in the node group's environment. When you select a parameter, the **Value** field is automatically populated with the inherited or default value.

- Optional: If necessary, change the parameter's default **Value**.

Related information

[Tips for specifying parameter and variable values](#) on page 447

Parameters and variables can be structured as JSON, but, if they can't be parsed as JSON, they're treated as strings.

Set variables

Variables you set in the console become top-scope variables available to all Puppet manifests.

Important: You can structure variables as JSON, but, if they can't be parsed as JSON, they're treated as strings.

- In the console, click **Node groups** and select the node group you want to set a variable for.
- On the **Variables** tab, enter the name of the variable in the **Key** field, and enter the value you want to assign to the variable in the **Value** field.
- Click **Add variable** and commit changes.

Related information

[Tips for specifying parameter and variable values](#) on page 447

Parameters and variables can be structured as JSON, but, if they can't be parsed as JSON, they're treated as strings.

Tips for specifying parameter and variable values

Parameters and variables can be structured as JSON, but, if they can't be parsed as JSON, they're treated as strings.

You can use these data types and syntax to specify parameters and variables:

- Strings (for example, "centos"): You can use variable-style syntax, which interpolates the result of referencing a fact (for example, "I live at \$ipaddress."), or expression-style syntax, which interpolates the result of evaluating the embedded expression (for example, \${os"release"}).

Important: Strings must be double-quoted, because single quotes aren't valid JSON.

Tip: In the console, to provide a value containing a dollar sign that is not part of variable syntax (such as password hashes), you must use a backslash to escape each dollar sign and disable interpolation. For example, the password hash \$1\$nnkkFwEc\$saafFMXYaUVfKrDV4FLCm0/ must be escaped as \\$1\\$nnkkFwEc\\$saafFMXYaUVfKrDV4FLCm0/.

- Booleans (for example, true or false).
- Numbers (for example, 123).
- Hashes (for example, { "a" : 1}).

Important: You must use colons, not hash rockets.

- Arrays (for example, ["1" , "2 . 3"]).

Variable-style syntax

Variable-style syntax uses a dollar sign (\$) followed by a Puppet fact name, such as: "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.

Restriction: The \$pe_node_groups endpoint variable cannot be interpolated when used as a classifier in class variable values.

You can't use indexing 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 and followed by literally `[count]`. After interpolation, the final value is `{"count" => 4, "physicalcount" => 1}[count]`.

Restriction: 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 (such as `${ $os[release] }`), refer to trusted facts (such as `"My name is ${trusted[certname]}"`), or delimit fact names from strings (such as `"My ${os} release"`).

For example, this expression-style syntax accesses an operating system's full release number: `${ $os["release"] }`

Expression-style syntax uses these elements in this order:

1. An initial dollar sign and curly brace: `$ {`
2. A legal Puppet fact name preceded by an optional dollar sign.
3. Any number of index expressions. Quotation marks around indices are optional unless the index string contains spaces or square brackets.
4. A closing curly brace: `}`

Indices in expression-style syntax can be used to refer to trusted facts or to access individual fields in structured facts. Use strings in an index to access keys in 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 a zero-indexed integer.

These are examples of legal expression-style interpolation:

- `${os}`
- `${$os}`
- `${$os[release]}`
- `${$os['release']}`
- `${$os["release"]}`
- `${$os[2]}` (Being zero-indexed, this accesses the value of the third key-value pair in the `os` hash)
- `${$osrelease}` (This 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). These are 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 trusted facts must be interpolated using expression-style syntax. For example, the `certname` trusted fact is expressed as `"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.

Restriction: Regular expressions, resource references, and other keywords (such as `undef`) are not supported.

View nodes in a node group

You can view all nodes that currently match the rules specified for a node group.

1. In the console, click **Node groups** and select the node group you want to view.
2. Click **Matching nodes**.

The page displays the total number of nodes currently belonging to the group (referred to as *matching nodes*) and a list of the matching nodes' names.

If you use rules to [Dynamically add nodes to a node group](#) on page 443, matching nodes are determined by facts collected during the nodes' last Puppet runs, and the matching nodes list is updated when dynamic rules are added, deleted, and edited. Because of [How node group inheritance works](#) on page 440, nodes must match the rules in ancestor node groups, as well as the rules of the current node group, in order to appear on the matching nodes list.

If you [Statically add nodes to a node group](#) on page 442, the *pinned nodes* belong to the group regardless of other rules.

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 edit a node group to change the name, description, parent node group, environment, or environment group setting. You can delete node groups that have no children.

1. In the console, click **Node groups**, and select a node group.
2. Click one of these links (located in the upper-right area of the **Node group details** page):
 - **Edit node group metadata:** Enables edit mode so you can modify the node group's settings. Refer to [Create node groups](#) on page 441 for information about node group settings.
 - **Remove node group:** Deletes the node group. You must confirm the deletion. You can only delete node groups that are not parents of other node groups.
3. Commit changes.

Remove nodes from a node group

To remove nodes from a node group, you must either unpin the node or delete (or change) the dynamic rule that added the node.

Important: When a node is removed from a node group, the node is no longer classified with the classes declared in that node group. However, resources 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 the group's nodes, the Apache package **is not** removed from the node when the node no longer belongs to the node group.

1. In the console, click **Node groups**, select the node group you want to remove nodes from, and go to the **Rules** tab.
2. To remove nodes added to a group by [dynamic node group rules](#), determine which rules are adding the nodes, and:
 - Click **Remove** to remove a single rule. This also removes any other nodes matching this rule, if they do not match any other rules.
 - Click **Remove all rules** to remove all dynamic node group rules from the group. This removes **all** dynamically-added nodes from the group.
 - Change one or more rules so they exclude the nodes you do not want in the group. This might also add or remove other nodes from the group, depending on how specific the rule is.

3. To remove [pinned nodes](#), in the **Certname** table:

- Click **Unpin** to unpin an individual node.
- Click **Unpin all pinned nodes** to unpin all pinned nodes from the node group. This **does not** remove dynamically-added nodes. If a pinned node also falls under one of the dynamic node group rules, the node remains in the node group by virtue of the dynamic node group rule.

Tip: To unpin a node from all groups it's pinned to, use the `unpin-from-all` command endpoint. This does not remove a dynamically-added node – This command only removes manually [pinned nodes](#).

4. Commit changes.

Related information

[POST /v1/commands/unpin-from-all](#) on page 543

Unpin one or more specific nodes from all node groups they're pinned to. Unpinning has no effect on nodes that are assigned to node groups via dynamic rules.

Remove classes from a node group

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.
- Click **Remove all classes** to remove all classes from the node group.

Tip: If a class in the node group's class list is crossed out, the class has been deleted from your Puppet code. For example, the module containing the class was uninstalled.

3. Commit changes.

Edit or remove class parameters

Change class parameters assigned to node groups.

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 modify:

- **Edit:** Edit the parameter. For more information, refer to [Define data used by node groups](#) on page 446.
- **Remove:** Remove the parameter.

3. Commit changes.

Edit or remove variables

Change variables assigned to node groups.

1. In the console, click **Node groups**, and select a node group.

2. On the **Variables** tab, select an option:

- **Edit:** Edit the variable. For more information, refer to [Define data used by node groups](#) on page 446.
- **Remove:** Remove the variable.
- **Remove all variables:** Remove all variables assigned to the node group.

3. Commit changes.

Environment-based testing

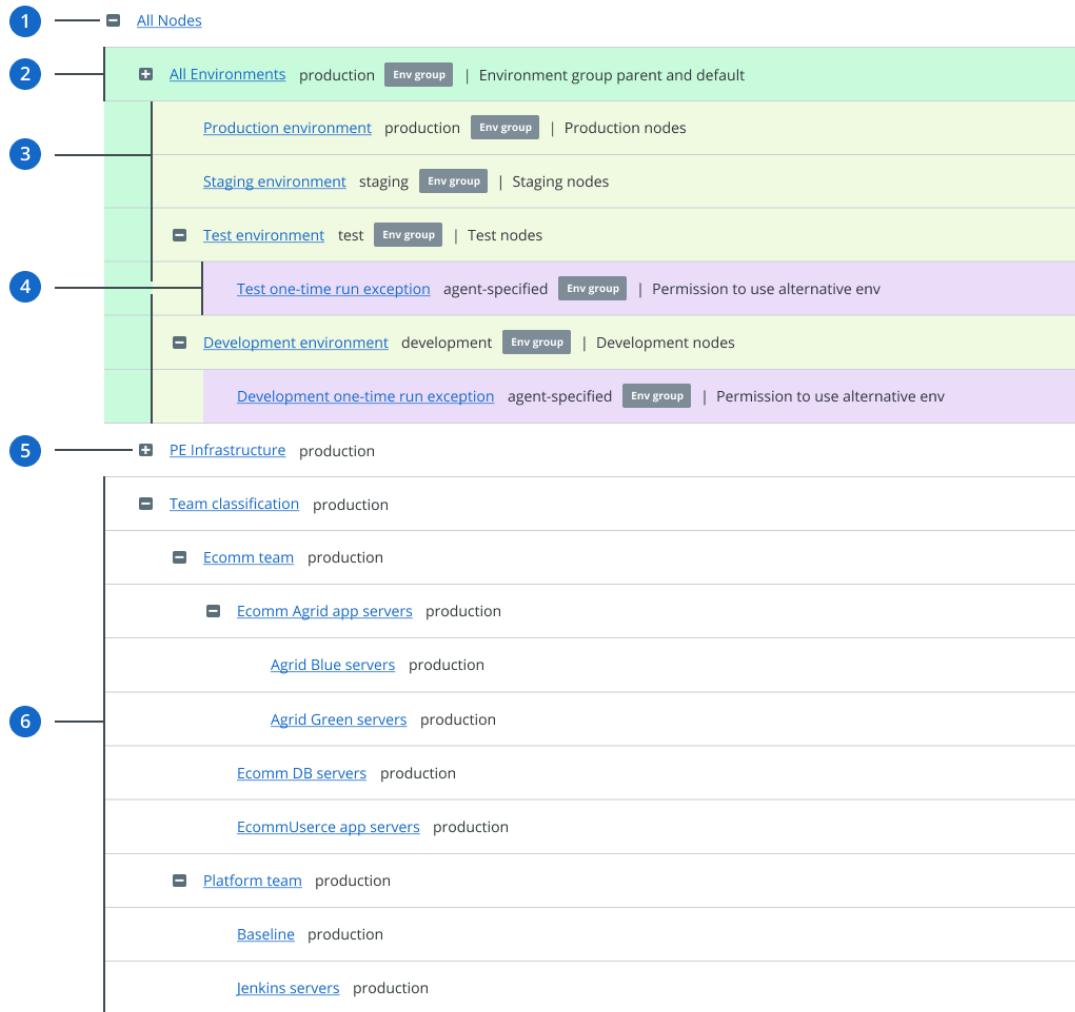
The environment-based testing workflow lets you test new code before pushing it to production.

Setting up environment-based testing requires a specific node group hierarchy. Before setting up environment-based testing, make sure you understand [Environment versus classification node groups](#) on page 441 and how to [Create node groups](#) on page 441.

The basic node group hierarchy for environment-based testing is as follows:

1. All Nodes (root node group of all node groups)
 - a. All Environments (parent of all environment node groups)
 1. Production environment node group
 2. Staging environment node group
 3. Test environment node group
 - a. Test environment one-time run exception node group (Used to [Test code with canary nodes and alternate Puppet environments](#) on page 454)
 4. Development environment
 - a. Development environment one-time run exception node group (Used to [Test code with canary nodes and alternate Puppet environments](#) on page 454)
 - b. PE Infrastructure node group
 - c. Classification node groups and their children (Child groups can be used to [Test and promote a parameter](#) on page 453 or [Test and promote a class](#) on page 454)

The following screenshot and table explain the environment-based testing node-group hierarchy components:



Call-out label	Group name or category	Description
1	All Nodes	Root node group of all node groups. Its direct children are the All Environments node group, the PE Infrastructure node group, and the top-level classification node groups.
2	All Environments	Each environment node group must be a child 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, create it.

Call-out label	Group name or category	Description
3	Various environment node groups, each being direct children of All Environments	Environment node groups specify the Puppet environment that your nodes belong to. Nodes can belong to only one environment node group, which determines the node's default environment.
4	Optional <i>one-time run exception</i> group, which is a direct child of a specific environment node group	<p>Used to Test code with canary nodes and alternate Puppet environments on page 454. This group acts as a gatekeeper, and, when present, permits nodes in the parent environment node group to temporarily use Puppet environments other than their normal default environment.</p> <p>This is the only condition where it is possible for an environment node group to have a child group.</p>
5	PE Infrastructure node group (and its child groups)	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	Various classification node groups and their children	Classification node groups apply classes and configuration data to nodes. Nodes can belong to multiple classification node groups. Child classification node groups can be used Test and promote a parameter on page 453 or Test and promote a class on page 454.

Test and promote a parameter

Use an environment-based testing workflow to test and promote a parameter.

1. [Create a classification node group](#). Set the **Environment** as the test environment, and set the **Parent** as the equivalent production environment classification node group.
By creating a test group associated with the test environment, the node classifier validates parameters against the test environment, rather than the production environment.
2. In your new test group, [set a parameter](#) you want to test.
Because you're setting the parameter on a child group, the test parameter overrides the value set by the parent group.
3. If you're satisfied with the results of the test, manually [edit the class parameter](#) in the parent group to apply the change to the production environment.
4. Delete the child group you used for testing.

Test and promote a class

Use an environment-based testing workflow to test and promote a class.

1. [Create a classification node group](#). Set the **Environment** as the test environment, and set the **Parent** as the equivalent production environment classification node group.
By creating a test group associated with the test environment, the node classifier validates classes and parameters against the test environment, rather than the production environment.
2. In your new test group, [declare a class](#) you want to test.
3. If you're satisfied with the results of the test, apply the change to the production environment by promoting the code containing the new classes to production (if necessary) and declaring the class in the parent node group.
4. Delete the child group you used for testing.

Test code with canary nodes and alternate Puppet environments

You can test new code using one-time Puppet agent runs on specific canary nodes.

Before you begin

You must [Create environment node groups](#) on page 441 other than **All Environments** in which to test code, such as development or test environment node groups.

To enable the canary workflow, create a *one-time run exception* child group under each environment node group in which you want to test code. This group matches nodes on the fly when you run Puppet with the environment specified.

For example, the following steps create a *one-time run exception* environment group as a child of the **Development environment** node group. You can create a similar group for any environment you want to test.

1. In the console, click **Node groups**, and click **Add group**.
2. Create the *one-time run exception* environment node group with these options:
 - **Name:** <ENVIRONMENT> *one-time run exception*, such as Development *one-time run exception*
 - **Parent:** Whichever environment you are testing, such as the **Development environment**
 - **Environment:** Select **agent-specified**
 - **Environment group:** Select this option.
3. Click **Add**.
4. [Dynamically add nodes to a node group](#) on page 443 by creating a rule to match nodes to this group if they request a Puppet environment other than their default environment:
 - **Fact:** `agent_specified_environment`

Tip: This fact name doesn't autocomplete. You must manually type it.

- **Operator:** `~`
- **Value:** `.+`

This rule matches any node from the parent environment node group that requests to use a non-default environment, through either the `--environment` flag on the command line or the `environment` setting in `puppet.conf`. For other rules you might use, refer to [Sample rules for one-time run exception groups](#) on page 455.

- On any node in the parent environment node group (such as the **Development environment** group), run the following code:

```
puppet agent -t --environment <ENVNAME>
```

Important: <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 **one-time run exception** group's rule 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>		.+
Permit RHEL nodes in the parent environment group to use any Puppet environment	<code>agent_specified_environment</code>		.+
	<code>facts.os.family</code>	=	RedHat
Permit any nodes in the parent environment group to use any Puppet environment except production	<code>agent_specified_environment</code>		.+
	<code>agent_specified_environment</code>		production
Permit any nodes in the parent environment group to use any Puppet environment that begins with the prefix "feature_	<code>agent_specified_environment</code>		^feature_.+

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

The **PE Infrastructure** node group is the parent to all other infrastructure node groups. This node group contains data, such as the service hostnames, service ports, and database info (excluding passwords).



CAUTION:

It's important to correctly configure the `puppet_enterprise` class in the **PE Infrastructure** node group. This class' parameters impact the behavior of all other preconfigured node groups that use classes starting with `puppet_enterprise::profile`. Incorrectly configuring this class can cause a service outage.

Don't remove the **PE Infrastructure** node group. Removing this node group disrupts communication between all of your infrastructure nodes.

Classes

`puppet_enterprise`: Sets the default parameters for all child infrastructure node groups

Matching nodes

Nodes are not pinned to this node group.

The **PE Infrastructure** node group is the parent to all other infrastructure node groups, such as **PE Master**. The **PE Infrastructure** node group's only purpose is to set classifications that are inherited by all child infrastructure node groups.

Never pin nodes directly to the **PE Infrastructure** node group. Instead, pin nodes to children of this group.

The following table describes the `puppet_enterprise` class parameters set on the **PE Infrastructure** node group (and that are inherited by child infrastructure node groups).

Tip: <YOUR HOST> is your primary server's certname. To find the certname run: `puppet config print certname`

Parameter	Value
<code>certificate_authority_host</code>	"<YOUR HOST>"
<code>console_host</code>	"<YOUR HOST>"
<code>console_port</code>	443 or another port number. Only change this if you changed the PE console service port number from the default of 443.
<code>database_host</code>	"<YOUR HOST>"
<code>database_port</code>	5432 or another port number. Only change this if you changed the PostgreSQL database port from the default of 5432.
<code>database_ssl</code>	<code>true</code> if you're using PE-installed PostgreSQL. <code>false</code> if you're using your own PostgreSQL.

Parameter	Value
pcp_broker_host	"<YOUR HOST>"
puppet_master_host	"<YOUR HOST>"
puppetdb_database_name	"pe-puppetdb"
puppetdb_database_user	"pe-puppetdb"
puppetdb_host	["<YOUR HOST>"]
puppetdb_port	[8081] or another port number. Only change this if you changed the PuppetDB port number from the default of 8081. For example, if The PuppetDB default port conflicts with another service on page 863.

Related information

[Database configuration parameters](#) on page 130

These parameters and values are supplied for Puppet Enterprise (PE) databases.

PE Certificate Authority node group

This node group is used to manage the certificate authority (CA).

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. To avoid issues, don't set the `client_allowlist` parameter of the `puppet_enterprise::profile::certificate_authority` class in this node group. Instead, to grant certificates access to the CA API without listing individual certificate names, use the `"pp_cli_auth": "true"` certificate extension. For instructions, see [Puppet-specific registered IDs](#).

PE Master node group

This node group is used to manage the primary server.

Classes

- `puppet_enterprise::profile::master` — manages the primary server service

Matching nodes

On a new install, the primary server is pinned to this node group.

PE Compiler node group

This node group is a subset of the **PE Master** node group used to manage compilers running the PuppetDB service.

Classes

- `puppet_enterprise::profile::master` — manages the primary server service
- `puppet_enterprise::profile::puppetdb` — manages the PuppetDB service

Matching nodes

Compilers running the PuppetDB service are automatically added to this node group.

Notes

Don't add additional nodes to this node group.

PE Orchestrator node group

This node group is used to manage the PE orchestration services configuration, which includes things like task concurrency limits, the PCP broker timeout, and how many JRubies can run in the orchestrator at one time.

Classes

`puppet_enterprise::profile::orchestrator` — manages PE orchestration services

Matching nodes

On a new install, the primary server is pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE PuppetDB node group

This node group is used to manage nodes running the PuppetDB service. If the node is also serving as a compiler, it's instead classified in the **PE Compiler** node group.

Classes

`puppet_enterprise::profile::puppetdb` — manages the PuppetDB service

Matching nodes

PuppetDB nodes that aren't functioning as compilers are pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE Console node group

This node group is used to manage the console.

Classes

- `puppet_enterprise::profile::console` — manages the console, node classifier, and RBAC
- `puppet_enterprise::license` — manages the PE license file for the status indicator

Matching nodes

On a new install, the console server node is pinned to this node group.

Notes

Don't add additional nodes to this node group.

PE Agent node group

This node group is used to manage the configuration of agents.

Classes

`puppet_enterprise::profile::agent` — manages your agent configuration

Matching nodes

All managed nodes are pinned to this node group by default.

PE Infrastructure Agent node group

This node group is a subset of the **PE Agent** node group used to manage infrastructure-specific overrides.

Classes

`puppet_enterprise::profile::agent` — manages your agent configuration

Matching nodes

All nodes used to run your Puppet infrastructure and managed by the PE installer are pinned to this node group by default, including the primary server, PuppetDB, console, and compilers.

Notes

You might want to manually pin to this group any additional nodes used to run your infrastructure, such as compiler load balancer nodes. Pinning a compiler load balancer node to this group allows it to receive its catalog from the primary server, rather than the compiler, which helps ensure availability.

PE Database node group

This node group is used to manage the PostgreSQL service.

Classes

- `puppet_enterprise::profile::database` — manages the PE-PostgreSQL service

Matching nodes

The node specified as `puppet_enterprise::database_host` is pinned to this group. By default, the database host is the PuppetDB server node.

Notes

Don't add additional nodes to this node group.

PE Patch Management node group

This is a parent node group for nodes under patch management. Create child node groups based on your needs.

Classes

`pe_patch` — enables patching on nodes.

Matching nodes

There are no nodes pinned to this group. **PE Patch Management** is a parent group for node groups under patch management. You can create node groups with unique configurations based on your patching needs.

Notes

Don't add additional nodes to this node group, only add node groups.

Related information

[Create a node group for nodes under patch management](#) on page 576

Create a node group for nodes you want to patch in Puppet Enterprise (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 Puppet Enterprise (PE) to manage your Windows configurations, including controlling services, creating local group and user accounts, and performing basic management tasks with modules from the Forge.

Make sure you understand how to run [Commands with elevated privileges](#) on page 30.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Related information

[Using example commands](#) on page 28

These guidelines can help you understand and customize the example commands you'll find in the Puppet Enterprise (PE) docs.

[Install Windows agents](#) on page 154

There are many ways you can install agents on Windows nodes, including PowerShell scripts, the Puppet Enterprise (PE) console, the MSI installer, and the `msiexec` command.

[Setting agent versions](#) on page 200

Usually, you want your agent nodes to run the same agent version as the primary server; however, if absolutely necessary, agent nodes can run a different, but compatible, version.

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.

You'll create a simple manifest file and use it to perform some common actions.

Practice tasks

You'll encounter these tasks as part of other tasks throughout the Puppet Enterprise (PE) documentation. We've provided additional explanation here for your reference.

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, such as Visual Studio Code (VSCode), to create manifest files. Puppet has a VSCode extension that supports syntax highlighting of the Puppet language. Editors like Notepad++ or Notepad don't highlight Puppet syntax, but you can use them to create manifests.

Manifest files are written in Puppet code, a domain specific language (DSL), and define the desired state of system resources, such as file, users, and packages. Puppet compiles these text-based manifests into catalogs, and uses those catalogs 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 code 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, the resource type is `file`.
- The vvalue before the `:` is the resource title. In this example, the title is `C:\\Temp\\foo.txt`. The file resource uses the title to determine where to create the file on disk. Resource titles must always be unique within a given manifest.
- The `ensure` parameter is set to `present` to make sure the file exists on disk and create the file if it doesn't already exist. 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.

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**.

Note these details about the Puppet command prompt:

- Several important batch files live in the current working directory, which is at `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.

Validate your manifest with `puppet parser validate`

You can validate that a manifest's syntax is correct by using the `puppet parser validate` command.

1. Open the Puppet command prompt and check your syntax by running: `puppet parser validate c:\myfiles\file.pp`
If the manifest has no syntax errors, the tool outputs nothing.
2. To preview the error output, edit your sample manifest file to remove the `:` after the resource title, and run `puppet parser validate c:\myfiles\file.pp` again to return the error response:

```
Error: Could not parse for environment production: Syntax error at
'ensure' at c:/myfiles/file.pp:2:3
```

Simulate a Puppet run with `--noop`

Puppet has a switch that you can use to test if manifests make your intended changes. This is referred to as non-enforcement mode or no-op mode.

To simulate changes, run `puppet apply c:\myfiles\file.pp --noop` in the command prompt and observe the result:

```
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. In the above example, 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`

If you are satisfied with the outcome of a no-op run, you can start enforcing the changes with the `puppet apply` command.

Run `puppet apply` with the desired manifest file, such as: `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:

1. 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.'
}
```

2. Apply the manifest by running: `puppet apply c:\myfiles\file.pp`

3. Open c:\Temp\foo.txt and notice that Puppet changed the file's contents.

Applying the manifest again (with `puppet apply c:\myfiles\file.pp`) results in no changes to the system, because the file already exists in the desired state, thereby demonstrating that Puppet behaves idempotently.

Many of the samples in the 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.

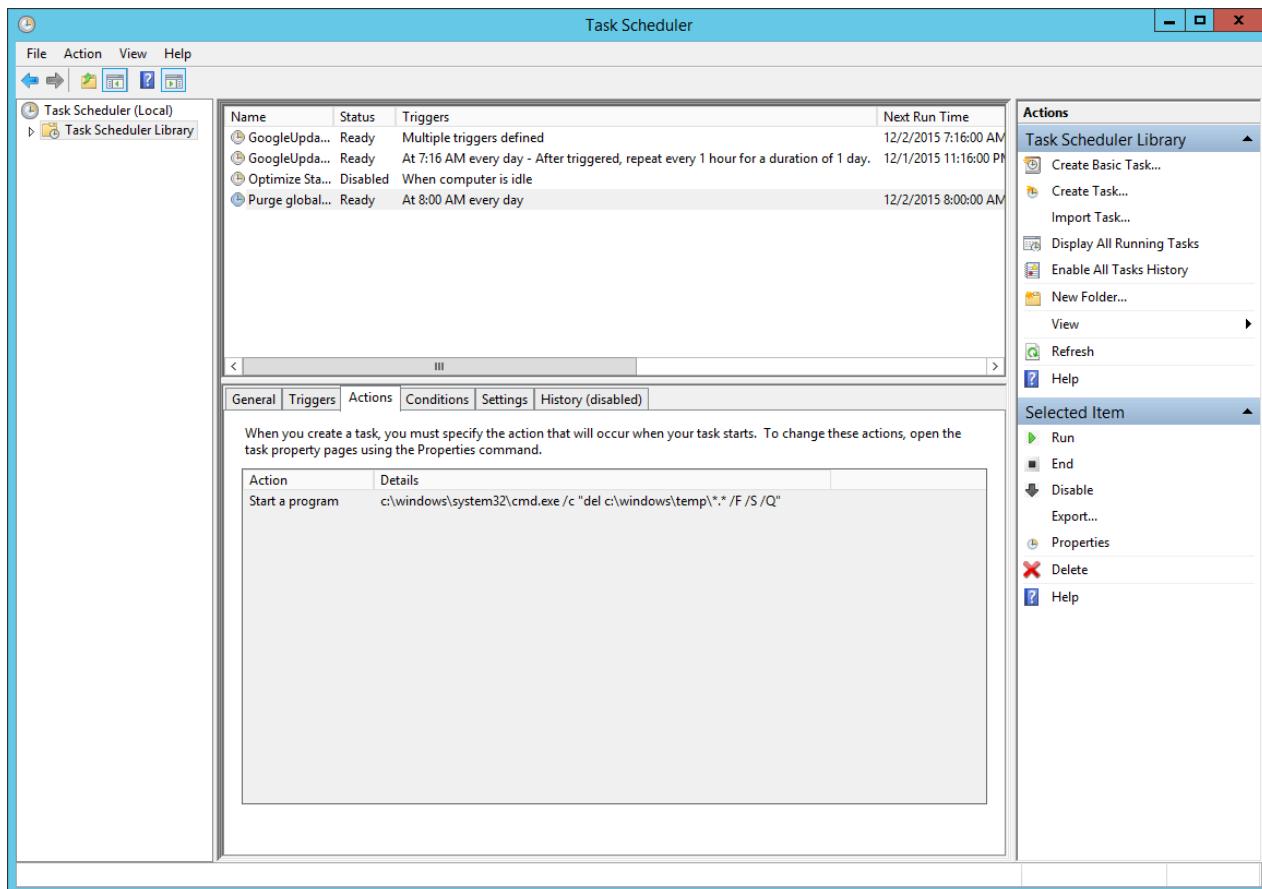
Note: If you need to run an ad-hoc task in the PE console or on the command line, see [Running tasks in PE](#) on page 615.

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:

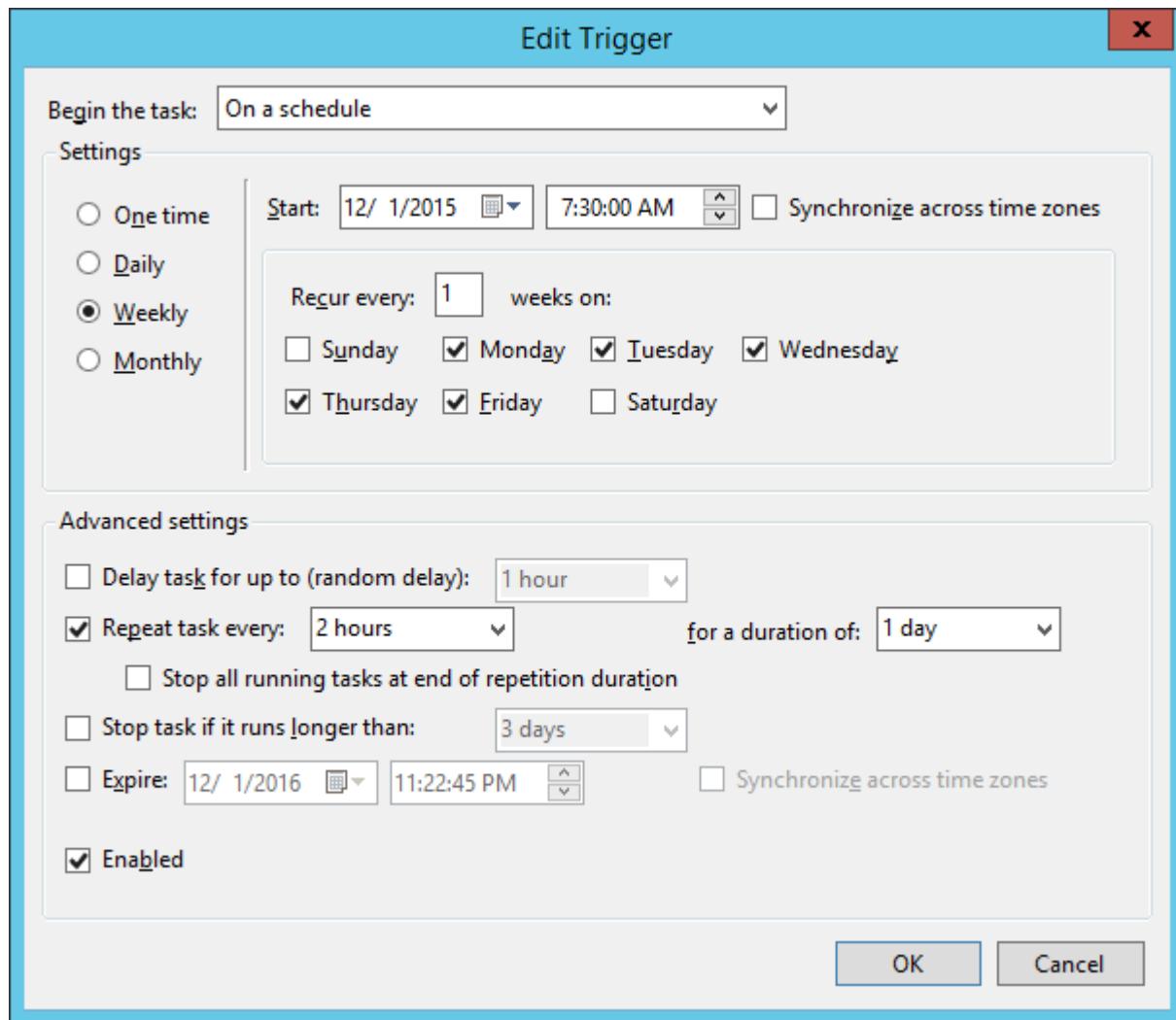


Example

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 => '<PASSWORD>'
}
```

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

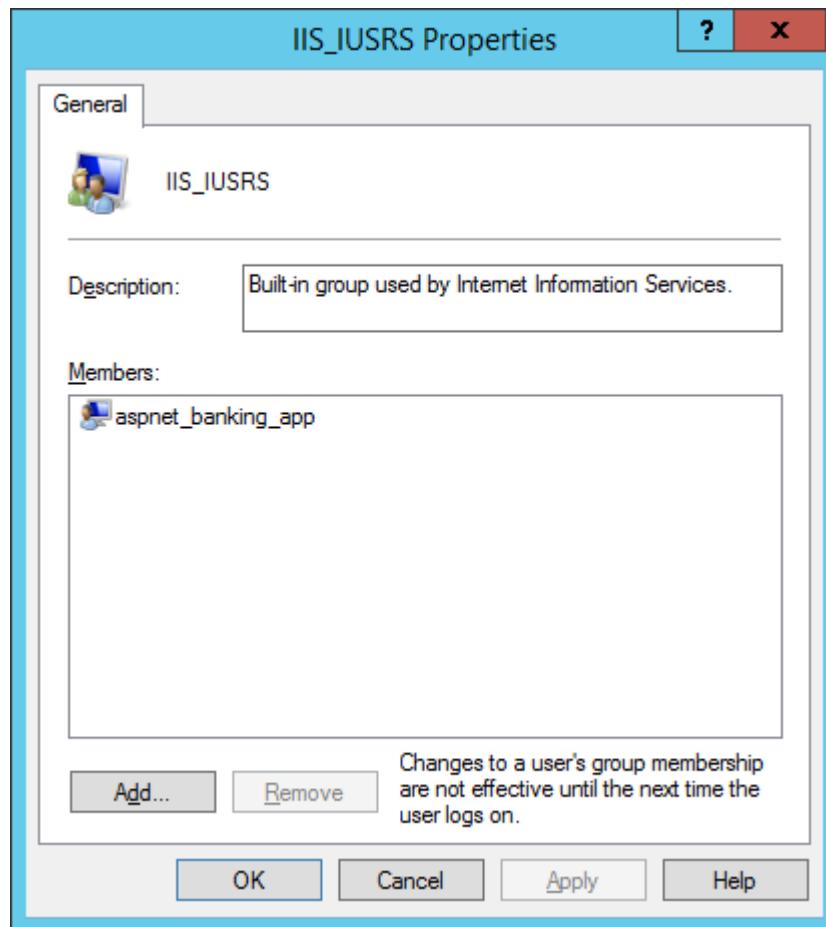
You might not always want to use the default user for an application, you can use Puppet to create users for other applications, like ASP.NET.

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.

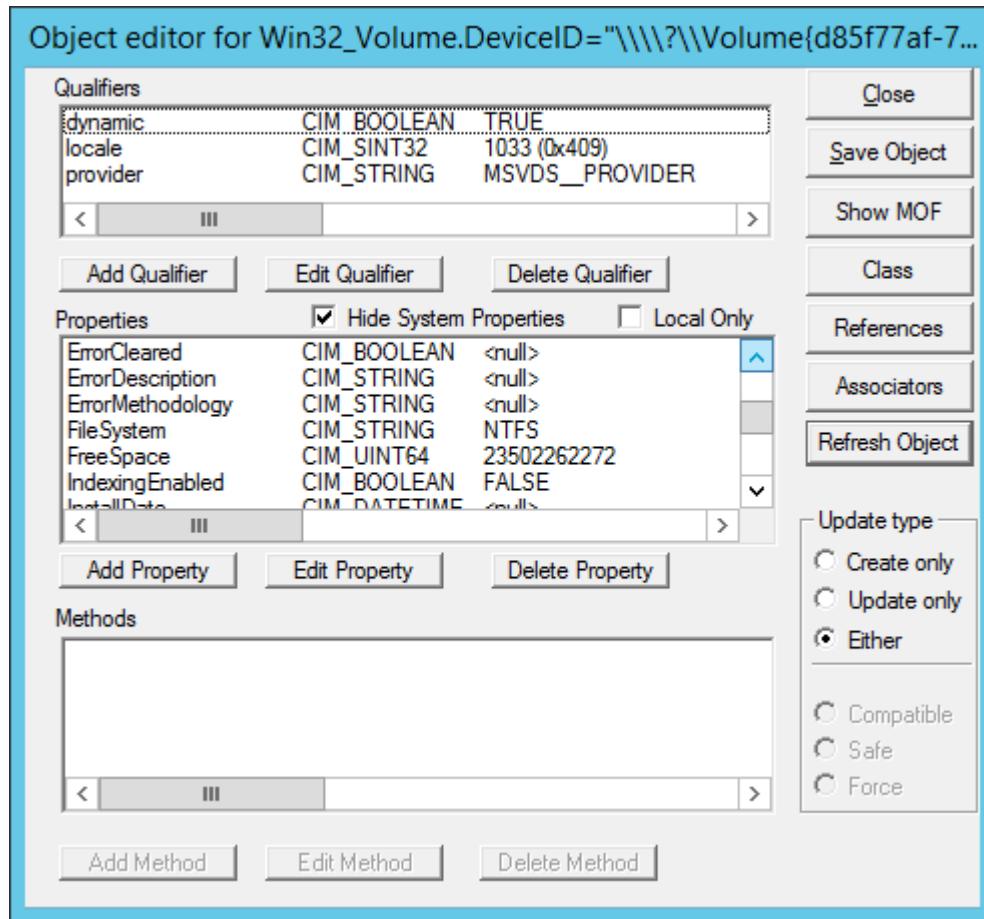
A common Windows maintenance tasks is to disable Windows drive indexing, because it can negatively impact disk performance on servers.

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

- Run this code to create your first Puppet managed permission. Then, save it as perms.pp

```
file{ 'c:/tempperms':
  ensure => directory,
}

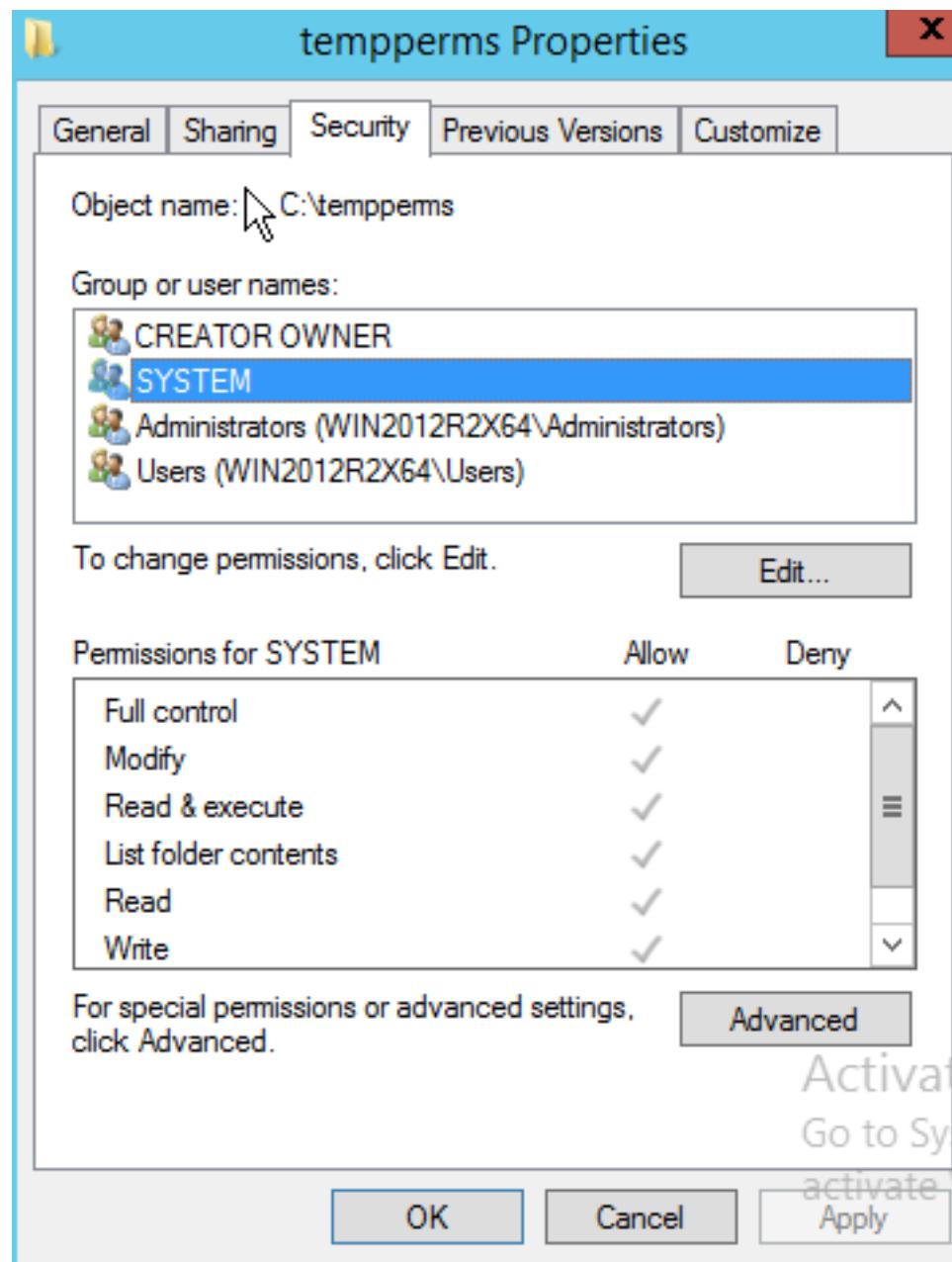
# By default, the acl creates an implicit relationship to any
# file resources it finds that match the location.
acl { 'c:/tempperms':
  permissions => [
    {identity => 'Administrators', rights => ['full']},
    {identity => 'Users', rights => ['read', 'execute']}
  ],
}
```

- 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.
- To apply the manifest, type `puppet apply c:\<FILE PATH>\perms.pp`

Your output should look similar to:

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.12 seconds
Notice: /Stage[main]/Main/File[c:/tempperms]/ensure: created
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions
changed [
] to [
  { identity => 'BUILTIN\Administrators', rights => ["full"] },
  { identity => 'BUILTIN\Users', rights => ["read", "execute"] }
]
Notice: Applied catalog in 0.05 seconds
```

4. Review the permissions in your Windows UI. In Windows Explorer, right-click **tempperms** and click **Properties**. Then, click the **Security** tab. It should appear similar to the image below.



5. Optional: It might appear that you have more permissions than you were hoping for here. This is because by default Windows inherits parent permissions. In this case, you might not want to do that. Adjust the acl resource to not inherit parent permissions by changing the `perms.pp` file to look like the below by adding `inherit_parent_permissions => false`.

```
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 seconds
Notice: /Stage[main]/Main/Acl[c:/tempperms]/permissions: permissions changed [
{ identity => 'BUILTIN\Administrators', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["read", "execute"] },
{ identity => 'NT AUTHORITY\SYSTEM', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["mask_specific"], mask => '4',
  child_types => 'containers' },
{ identity => 'CREATOR OWNER', rights => ["full"], affects =>
  'children_only' }
] to [
{ identity => 'BUILTIN\Administrators', rights => ["full"] },
{ identity => 'BUILTIN\Users', rights => ["read", "execute"] }
]
Notice: Applied catalog in 0.05 seconds
```

Puppet outputs a notice as it is removing each of the permissions.

12. Take a look at the output of `icacls` again with `icacls c:\tempperms`

```
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 examine existing registry settings.

This command is somewhat limited, like the `acl` module, in that it is restricted to only what is specified.

Keys are like file paths (directories), and values are like files that can have data and be of different types.

- To examine a registry key, open a command prompt and run:

```
puppet resource registry_key 'HKLM\Software\Microsoft\Windows'
```

```
registry_key { 'HKLM\Software\Microsoft\Windows\':
  ensure => 'present',
}
```

- To examine a registry value, run:

```
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',
}
```

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.

- 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',
}
```

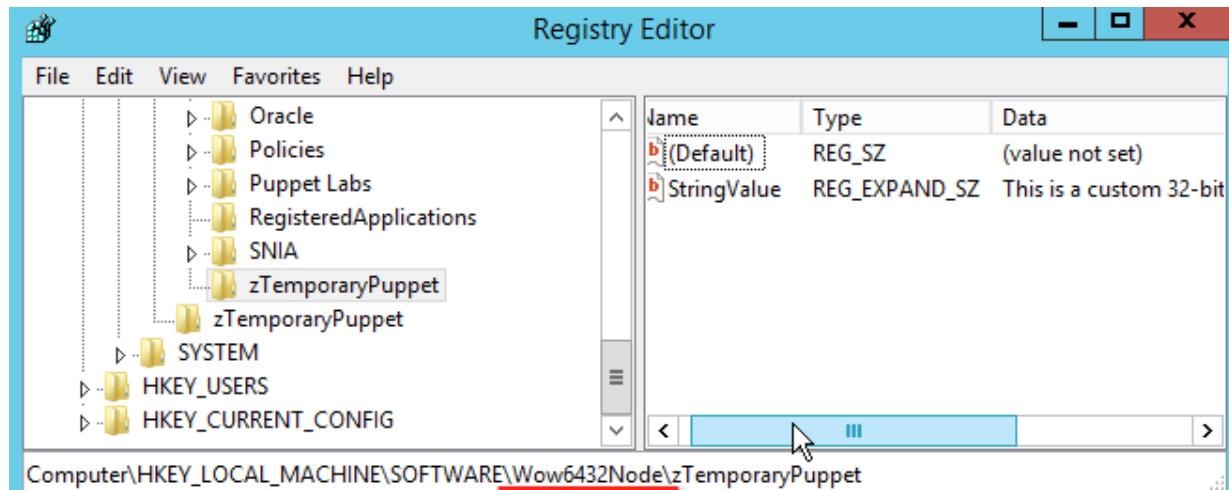
- Save the file as `registry.pp`.
- Validate the manifest. In the command prompt, run `puppet parser validate c:\<FILE PATH>\registry.pp`

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**.



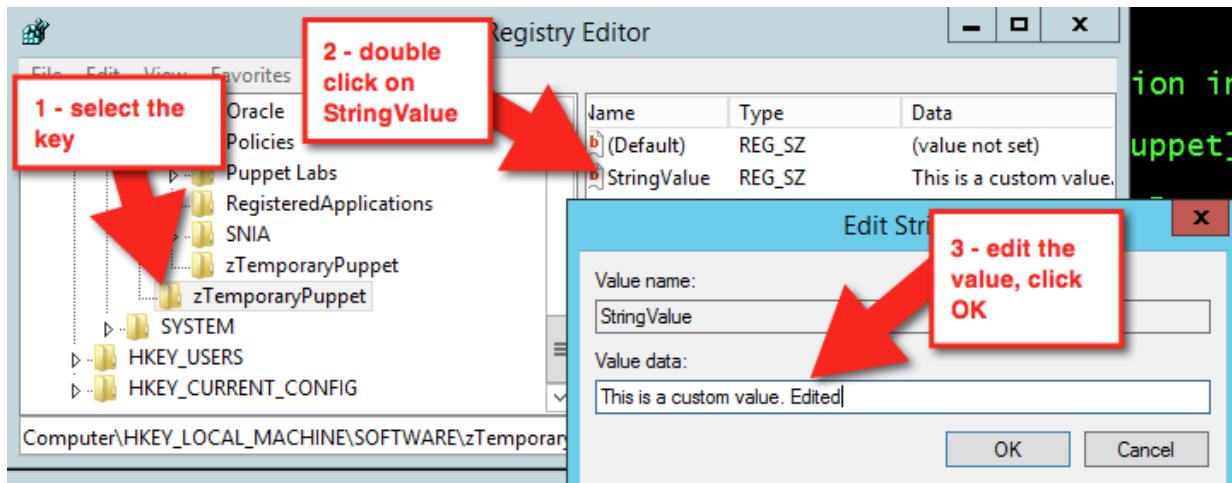
Note that the 32-bit keys were created under the 32-bit section of Wow6432Node for Software.

- 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:

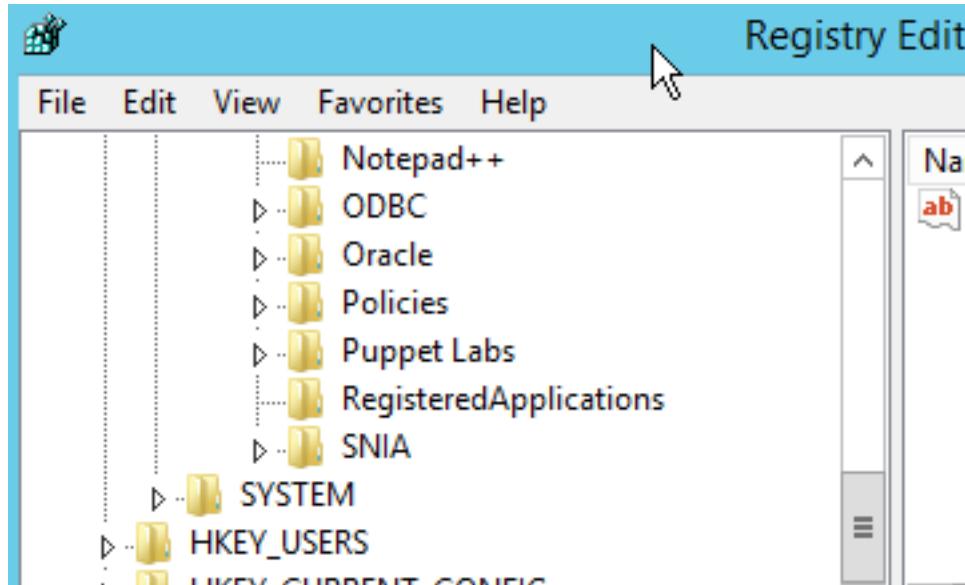
```
registry_key { 'HKLM\Software\zTemporaryPuppet':
  ensure => absent,
}

#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[HKEY_LOCAL_MACHINE\Software\zTemporaryPuppet]/
ensure: removed
Notice: /Stage[main]/Main/Registry_key[32:HKEY_LOCAL_MACHINE\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 repackage packages with the chocolatey module

Chocolatey is a package manager for Windows that is similar in design and execution to package managers on non-Windows systems. The `chocolatey` module is a Puppet Approved Module, so it's not eligible for Puppet Enterprise support services. The module has the capability to install and configure Chocolatey itself, and then manage software on Windows with Chocolatey packages.

View existing packages

Chocolatey has a custom provider for the package resource type, so you can use `puppet resource` to view existing packages.

In the command prompt, run `puppet resource package --param provider` | more

The additional provider parameter in this command outputs all types of installed packages that are detected by multiple providers.

Install Chocolatey

These steps are to install Chocolatey (choco.exe) itself. You use the chocolatey 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`

Make sure the output is similar to the following:

```
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'll likely have a `Chocolatey.nupkg` file located somewhere internally. In these cases, you can use the internal nupkg to install Chocolatey, such as:

```
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).

7. Next, open the * .nuspec file to view its contents and make any necessary changes.

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>notepadplusplus.commandline</id>
    <version>6.8.7</version>
    <title>Notepad++ (Portable, CommandLine)</title>
    <authors>Don Ho</authors>
    <owners>Rob Reynolds</owners>
    <projectUrl>https://notepad-plus-plus.org/</projectUrl>
    <iconUrl>https://cdn.rawgit.com/ferventcoder/chocolatey-packages/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.

8. Now you can navigate via the command prompt to the folder with the .nuspec file (from a Windows machine unless you've installed Mono and built choco.exe from source) and use choco pack. You can also be more specific and run choco pack <FILE PATH>\notepadplusplus.commandline.nuspec. The output should be similar to below.

```
Attempting to build package from 'notepadplusplus.commandline.nuspec'.
Successfully created package 'notepadplusplus.commandline.6.8.7.nupkg'
```

Normally you test on a system to ensure that the package you just built is good prior to pushing the package (just the * .nupkg) to your internal repository. This can be done by using choco.exe on a test system to install (choco install notepadplusplus.commandline -source %cd% - change %cd% to \$pwd in PowerShell.exe) and uninstall (choco uninstall notepadplusplus.commandline). Another method of testing is to run the manifest pointed to a local source folder, which is what you are going to do.

9. Create c:\packages and copy the resulting package file (notepadplusplus.commandline.6.8.7.nupkg) into it.

This won't actually install on this system since you just installed the same version from Chocolatey's community feed. So you need to remove the existing package first. To remove it, edit your chocolatey.pp to set the package to absent.

```
package { 'notepadplusplus.commandline':
  ensure  => absent,
  provider => chocolatey,
}
```

10. Validate the manifest with puppet parser validate path\to\chocolatey.pp. Apply the manifest to ensure the change puppet apply c:\path\to\chocolatey.pp.

You can validate that the package has been removed by checking for it in the package install location or by using choco list -lo.

11. Update the manifest (`chocolatey.pp`) to use the custom package.

```
package { 'notepadplusplus.commandline':
  ensure  => latest,
  provider => chocolatey,
  source   => 'c:\packages',
}
```

12. Validate the manifest with the parser and then apply it again. You can see Puppet creating the new install in the output.

```
Notice: Compiled catalog for win2012r2x64 in environment production in
0.79 seconds
Notice: /Stage[main]/Main/Package[notebookplusplus.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:\temp\packages\vagrant
Generating template to a file
  at 'C:\temp\packages\vagrant\vagrant.nuspec'
Generating template to a file
  at 'C:\temp\packages\vagrant\tools\chocolateyinstall.ps1'
Generating template to a file
  at 'C:\temp\packages\vagrant\tools\chocolateyuninstall.ps1'
Generating template to a file
  at 'C:\temp\packages\vagrant\tools\ReadMe.md'
Successfully generated vagrant package specification files
  at 'C:\temp\packages\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.

The screenshot shows the Windows Control Panel with the title 'Programs and Features'. Below it, a sub-menu bar shows 'All Control Panel Items' and 'Programs and Features'. A search bar on the right says 'Search Programs and...'. The main area is titled 'Uninstall or change a program' with the instruction 'To uninstall a program, select it from the list and then click Uninstall, Change, or Repair'. A table lists programs:

Name	Publisher
Vagrant	HashiCorp
Microsoft Visual C++ 2010 x86 Redistributable - 10.0....	Microsoft Corporation

 A red arrow points to the 'Vagrant' row in the list.

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
```

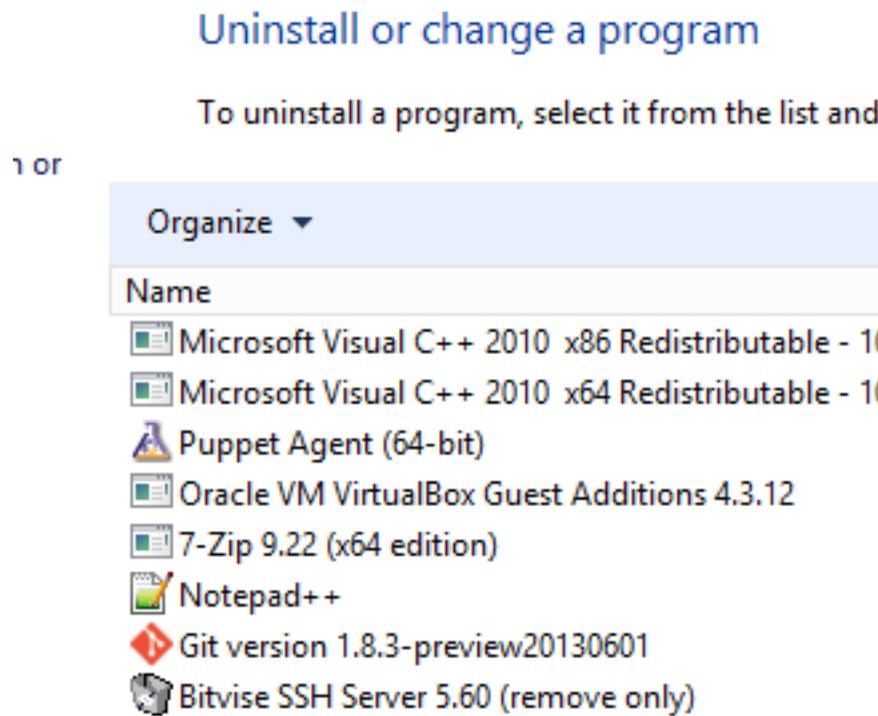
2. To remove Vagrant, edit your `chocolatey.pp` manifest to ensure `=> absent`. Then save and validate the file.

```
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!



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 486

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

- [Roles and profiles example](#) on page 490

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 493

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 510

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

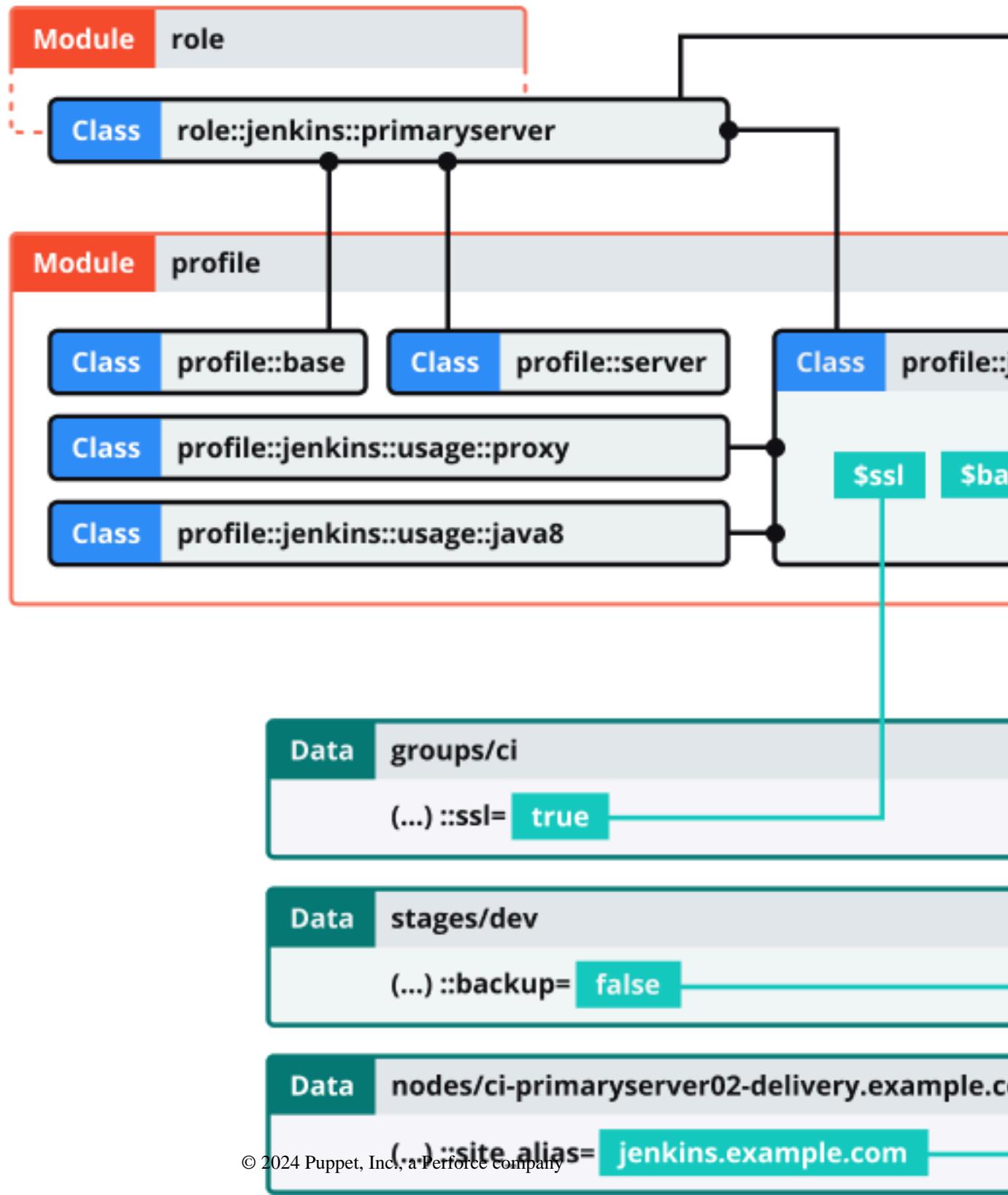
Roles and profiles are *two extra layers of indirection* between your node classifier and your component modules.

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



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::primaryserver {
    include profile::base
    include profile::server
    include profile::jenkins::primaryserver
}
```

- Each profile configures a layered technology stack, using multiple component modules and the built-in resource types. (In the diagram, `profile::jenkins::primaryserver` 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 primary server," it makes sense to write a role named `role::jenkins::primaryserver`. 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 controller servers with roles and profiles

Jenkins is a continuous integration (CI) application that runs on the JVM. The Jenkins controller 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 controller 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::controller`, located at `.../profile/manifests/jenkins/controller.pp`, and fill it with Puppet code.

```
# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/controller.pp
class profile::jenkins::controller (
  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 489

There are rules for writing profile classes.

[Methods for data lookup](#) on page 490

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 controllers, but most of them belong to the `ci` group.
- Our quality engineering department wants controllers 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 controllers 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-
controller01.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::controller::jenkins_port: '80'

# /etc/puppetlabs/code/environments/production/data/groups/ci.yaml
profile::jenkins::controller::java_dist: 'oracle-jdk8'
profile::jenkins::controller::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 controllers 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 controllers should include those classes as well.

```
class role::jenkins::controller {
  include profile::base
  include profile::server
  include profile::jenkins::controller
}
```

Related information

[Rules for role classes](#) on page 489

There are rules for writing role classes.

Assign the role to nodes

Finally, we assign `role::jenkins::controller` to every node that acts as a Jenkins controller.

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
```

To learn more about how to assign custom facts to individual nodes, visit https://puppet.com/docs/puppet/7/fact_overview.html.

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/controller.pp
class profile::jenkins::controller (
  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 489

There are rules for writing profile classes.

First refactor: Split out Java

We want to manage Jenkins controllers *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 controller 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::controller` 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::controller (
  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/controller.pp
class profile::jenkins::controller (
-  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::controller (
  },
}

- 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 controller, in MB.
if($::memoriesize_mb =~ Number and $::memoriesize_mb > 8192)
{
  # anything over 8GB we should keep max 4GB for OS and others
  $heap = sprintf('%.0f', $::memoriesize_mb - 4096)
} else {
  # This is calculated as 50% of the total memory.
  $heap = sprintf('%.0f', $::memoriesize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
  value => "-Xms${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 controllers that can't spare the extra memory. But because our production controllers 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::controller (
    # When not using the jenkins module's java version, install java8.
    unless $install_jenkins_java { include profile::jenkins::usage::java8 }
+
+ # Manage the heap size on the controller, in MB.
+ if($::memoriesize_mb =~ Number and $::memoriesize_mb > 8192)
+ {
+   # anything over 8GB we should keep max 4GB for OS and others
+   $heap = sprintf('%.0f', $::memoriesize_mb - 4096)
+ } else {
+   # This is calculated as 50% of the total memory.
+   $heap = sprintf('%.0f', $::memoriesize_mb * 0.5)
+ }
+ # Set java params, like heap min and max sizes. See
+ # https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
+ jenkins::sysconfig { 'JAVA_ARGS':
+   value => "-Xms${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::controller (
  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/controller.pp
class profile::jenkins::controller {
- 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 controllers, 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
$controller_config_dir   =
$profile::jenkins::params::controller_config_dir

file { '/var/run/jenkins': ensure => 'directory' }

# Because our account::user class manages the '${controller_config_dir}' directory
# as the 'jenkins' user's homedir (as it should), we need to manage
# `${controller_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 { "${controller_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::controller (
  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
+ $controller_config_dir   =
  $profile::jenkins::params::controller_config_dir
+
+ file { '/var/run/jenkins': ensure => 'directory' }
+
+ # Because our account::user class manages the '${controller_config_dir}' directory
+ # as the 'jenkins' user's homedir (as it should), we need to manage
+ # `${controller_config_dir}/plugins` here to prevent the upstream
+ # rtyler-jenkins module from trying to manage the homedir as the config
+ # dir. For more info, see the upstream module's `manifests/plugin.pp`
+ # manifest.
+ file { "${controller_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::controller (
    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 { "${controller_config_dir}/.ssh":
  ensure => directory,
  owner  => $jenkins_owner,
  group  => $jenkins_group,
  mode    => '0700',
}

file { "${controller_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 { "${controller_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 controllers 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::controller (
  Boolean                      $manage_plugins = false,
  # ...
) {
  # ...
  if $manage_plugins {
    include profile::jenkins::controller::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/controller/plugins.pp
class profile::jenkins::controller::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/controller.pp
class profile::jenkins::controller (
  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::controller (
    $jenkins_group      = $profile::jenkins::params::jenkins_group
    $controller_config_dir =
$profile::jenkins::params::controller_config_dir

+ if $manage_plugins {
+   # About 40 jenkins::plugin resources:
+   include profile::jenkins::controller::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 '${controller_config_dir}''
directory
@@ -69,4 +84,29 @@ class profile::jenkins::controller (
    value => "-Xms${heap}m -Xmx${heap}m -Djava.awt.headless=true
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -
Dhudson.model.DirectoryBrowserSupport.CSP=\\\"default-src 'self'; img-src
'self'; style-src 'self';\\\"",
}
+
# Deploy the SSH keys that Jenkins needs to manage its agent machines and
+ # access Git repos.
+ file { "${controller_config_dir}/.ssh":
+   ensure => directory,
+   owner  => $jenkins_owner,
+   group  => $jenkins_group,
+   mode    => '0700',
+ }
+
+ file { "${controller_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 { "${controller_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::controller (
  Boolean                      $backups_enabled = false,
  # ...
) {
  # ...
  if $backups_enabled {
    backup::job { "jenkins-data-{$::hostname}":
      files => $controller_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::controller (
  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/controller.pp
class profile::jenkins::controller (
  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::controller (
    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 controller 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 { "${controller_config_dir}/.ssh":
@@ -109,4 +120,29 @@ class profile::jenkins::controller (
    source => "puppet://${secure_server}/secure/delivery/id_rsa-
jenkins.pub",
}

+ # Back up Jenkins' data.
+ if $backups_enabled {
+   backup::job { "jenkins-data-${::hostname}":
+     files => $controller_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::controller {
  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 controller (profile::jenkins::controller)': }

if $site_alias {
  motd::register { 'jenkins-site-alias':
    content => @("END"),
                profile::jenkins::controller::proxy

    Jenkins site alias: ${site_alias}
    |-END
    order    => 25,
  }
}
```

The bulk of the work is handled by a new profile called `profile::jenkins::controller::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::controller::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::controller::proxy` is a "private" profile that belongs solely to `profile::jenkins::controller`. 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/controller.pp
class profile::jenkins::controller (
- 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::controller (
  # 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::controller (
  include profile::jenkins::controller::plugins
}

+ motd::register { 'Jenkins CI controller
(profile::jenkins::controller)': }
+
+ # 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::controller::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::controller::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::controller (
  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::controller`.

```

# /etc/puppetlabs/code/environments/production/site/profile/manifests/
jenkins/controller.pp
# Class: profile::jenkins::controller
#
# Install a Jenkins controller that meets Puppet's internal needs.
#
class profile::jenkins::controller (
  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
$controller_config_dir         =
$profile::jenkins::params::controller_config_dir

if $manage_plugins {
    # About 40 jenkins::plugin resources:
    include profile::jenkins::controller::plugins
}

motd::register { 'Jenkins CI controller (profile::jenkins::controller)': }

# 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::controller::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::controller::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 '${controller_config_dir}' directory
# as the 'jenkins' user's homedir (as it should), we need to manage
# `${controller_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 { "${controller_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 controller, in MB.
if($::memoriesize_mb =~ Number and $::memoriesize_mb > 8192)
{
  # anything over 8GB we should keep max 4GB for OS and others
  $heap = sprintf('%.0f', $::memoriesize_mb - 4096)
} else {
  # This is calculated as 50% of the total memory.
  $heap = sprintf('%.0f', $::memoriesize_mb * 0.5)
}
# Set java params, like heap min and max sizes. See
# https://wiki.jenkins-ci.org/display/JENKINS/Features+controlled+by
+system+properties
jenkins::sysconfig { 'JAVA_ARGS':
  value => "-Xms${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 controller 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 { "${controller_config_dir}/.ssh":
    ensure  => directory,
    owner   => $jenkins_owner,
    group   => $jenkins_group,
    mode    => '0700',
}

file { "${controller_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 { "${controller_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 => $controller_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::controller {
  include profile::base
  include profile::server
  include profile::jenkins::controller
}
```

Related information

[Rules for role classes](#) on page 489

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 controllers.

With granular roles, we'd have at least two Jenkins controller roles. A basic one:

```
class role::jenkins::controller {
  include profile::base
  include profile::server
  include profile::jenkins::controller
}
```

...and an RE-specific one:

```
class role::jenkins::controller::release {
  include profile::base
  include profile::server
  include profile::jenkins::controller
  include profile::jenkins::controller::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::controller::release {
  include profile::base
  include profile::server
```

```

include profile::jenkins::controller

if $facts['group'] == 'release' {
  include profile::jenkins::controller::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::controller {
  # Parent role:
  include role::server
  # Unique classes:
  include profile::jenkins::controller
}

class role::jenkins::controller::release {
  # Parent role:
  include role::jenkins::controller
  # Unique classes:
  include profile::jenkins::controller::release
}

```

In this example, we reduce boilerplate by having `role::jenkins::controller` include `role::server`. When `role::jenkins::controller::release` includes `role::jenkins::controller`, 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::controller` class could include both `profile::server` and `profile::base`, and you could manage a Jenkins controller server by directly assigning `profile::jenkins::controller` 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 controllers" group in the console and assign it the `profile::base`, `profile::server`, and `profile::jenkins::controller` classes, doing much the same job as our basic `role::jenkins::controller` 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 API requests](#) on page 514

Requests to the node classifier API must be well-formed HTTP(S) requests.

- [Groups endpoints](#) on page 516

The `groups` endpoints create, read, update, and delete groups.

- [Classes endpoint](#) on page 532

Use the `classes` endpoint to retrieve a list of all classes.

- [Classification endpoints](#) on page 533

The `classification` endpoints accepts a node name and a set of facts, and then return information about how the specified node is classified. The output can help you test your node group classification rules.

- [Commands endpoint](#) on page 543

Use the `commands` endpoint to unpin specified nodes from all node groups they're pinned to.

- [Environments endpoints](#) on page 545

Use the `environments` endpoints to retrieve the node classifier's environment data. The responses tell you which environments are available, whether a named environment exists, and which classes exist in a certain environment.

- [Nodes check-in history endpoints](#) on page 548

Use the `nodes` endpoints to retrieve records about nodes that have checked into the node classifier.

- [Group children endpoint](#) on page 551

Use the `group-children` endpoint to retrieve a list of node groups descending from a specific node group.

- [Rules endpoint](#) on page 555

Use the `rules` endpoint to translate a node group rule condition into PuppetDB query syntax.

- [Import hierarchy endpoint](#) on page 555

Use the `import hierarchy` endpoint to delete all existing node groups from the node classifier service and replace them with the node groups defined in the body of the request.

- [Last class update endpoint](#) on page 558

Use the `last-class-update` endpoint to retrieve the time that classes were last updated from the primary server.

- [Update classes endpoint](#) on page 558

Use the `update-classes` endpoint to trigger the node classifier to get updated class and environment definitions from the primary server.

- [Validation endpoint](#) on page 559

Use the `validation` endpoint to validate groups in the node classifier.

- [Node classifier API errors](#) on page 562

Learn about node classifier API error responses.

Related information

[API index](#) on page 33

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

Forming node classifier API 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.

Node classifier API requests must include a URI path following the pattern:

```
https://<DNS>:4433/classifier-api/<VERSION>/<ENDPOINT>
```

The variable path components derive from:

- **DNS**: Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).
- **VERSION**: Either v1 or v2. Refer to [Node classifier API v2](#) on page 563 for v2 endpoints.
- **ENDPOINT**: One or more sections specifying the endpoint, such as `groups` or `classes`. Some endpoints require additional sections, such as the [GET /v1/environments/<environment>/classes](#) on page 546 endpoint.

For example, you could use any of these paths to call the [GET /v1/classes](#) on page 532 endpoint:

```
https://$(puppet config print server):4433/classifier-api/v1/classes
https://localhost:4433/classifier-api/v1/classes
https://puppet.example.dns:4433/classifier-api/v1/classes
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Rule condition grammar

You can use rules to dynamically classify nodes into groups. When forming requests to endpoints that support rule definition, you must use proper rule condition grammar, such as:

```
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. You must use literal backslashes 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 (such as `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 and subsequent entries can be indices.

Regular facts start with "fact" (for example, ["fact", "architecture"]) and trusted facts start with "trusted" (for example, ["trusted", "certname"]).

Related information

[Configure the console](#) on page 230

After installing Puppet Enterprise (PE), you can change product settings to customize the PE console's behavior. You can configure many of these settings directly in the console.

Node classifier API authentication

You must authenticate node classifier API requests. You can do this using RBAC authentication tokens or with the list of allowed RBAC certificates.

Authenticating with tokens

You can make requests to the node classifier API using RBAC authentication tokens.

For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

This example uses a token generated with `puppet-access login` to call the [GET /v1/groups](#) on page 516 endpoint:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"

curl --insecure --header "$auth_header" "$uri"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

The example above uses the X-Authentication header to supply the token information. In some cases, such as with GitHub webhooks, you might need to supply the token in a token parameter. For example:

```
uri="https://$(puppet config print server):4433/classifier-api/v1/groups?
token=$(puppet-access show)"

curl --insecure "$uri"
```



CAUTION: Supplying the token as a token parameter is not as secure as using the X-Authentication method.

Authenticating with 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 by running: `sudo service pe-console-services reload`

To attach the certificate to a curl request, you must have the allowed certificate name and the private key to run the script. The certname in the request must match a certname in the allowlist file at `/etc/puppetlabs/console-services/rbac-certificate-allowlist`. 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):4433/classifier-api/v1/groups"
```

```
curl --header "$type_header" --cert "$cert" --cacert "$cacert" --key "$key"
GET "$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 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

Using pagination parameters

If your installation has a large number of groups, classes, nodes, node check-ins, or environments, then node classifier API GET requests might return an excessively large results.

To limit the number of items returned, you can append the `limit` and `offset` parameters to your request URI paths:

- `limit`: Set the maximum number of items allowed to be returned in the response. The value must be a non-negative integer.
- `offset`: Specify the number of items to skip from the beginning of the possible results. The value must be a zero-indexed, non-negative integer. The response begins returning results from the point specified, for example if `offset=10`, the response skips the first 10 results and starts the response with the 11th record.

For example, if you specify an offset of 20 with a limit of 10 (as shown in the example below) the first 20 records are skipped, and the response returns records 21 through 30:

```
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"
```

Supplying non-integer values for these parameters returns a `400 Bad request` response.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Groups endpoints

The `groups` endpoints create, read, update, and delete groups.

Each group belongs to an environment, applies classes (which can have class parameters) to nodes within the group, and match nodes based on the group's rules. Because groups are central to the classification process, there is a lot you can do with this endpoint.

GET /v1/groups

Retrieves a list of all node groups in the node classifier.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, you can append the `inherited` parameter to the URI path. If this parameter is set to any value besides `0` or `false`, the response includes the classes, class parameters, configuration data, and variables that each group inherits from its ancestor groups.

For example, this request does not specify the `inherited` parameter:

```
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"
```

Tip:

Use the following curl command to get a pared-down response that lists all node groups and their corresponding IDs. Node group IDs are useful when running tasks or plans.

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/groups"
curl --silent --header "$auth_header" "$uri" | jq -M -r '.[] | "\(.name)\n\(.id)"'
```

The response is more simplified than the full response to the standard GET `/v1/groups` request:

```
All Nodes 00000000-0000-4000-8000-000000000000
PE Master 07002034-c20f-44de-97d2-f72d03e481fb
Development one-time run exception 124a11d8-b912-45f0-9a6d-5ddd81aaa0ed
PE PuppetDB 289f176b-1c30-4e85-ad07-131e55f29354
PE Database 28b78e75-3b7e-464e-8f02-29a80b88fe02
Development environment 388f2eea-2f91-4ed7-8f84-93d8bf115ec5
PE Orchestrator 3f490039-395f-4c87-8dfb-f72d03e481fb
Production environment 43d438de-78da-4186-9405-e3f743989a5c
All Environments 6a10e0eb-ab6b-4ba7-b637-28fdf91ed659
PE Compiler 9cab6f77-f0cf-4c0e-b2ce-6aa6a2489c71
PE Infrastructure 9cd74d7e-6fb7-4d17-9cd8-e3f743989a5c
PE Patch Management aae9e4cd-fed5-4f07-8149-98a699a3b692
PE Certificate Authority d4065370-0cab-43cf-a4fa-93d8bf115ec5
PE Agent d4cf6659-6fc8-4419-b2a2-28fdf91jh9607
PE Console de97e269-4a9e-4f3d-a931-39993b0ef3f6
PE Infrastructure Agent e46543a6-61c6-49f2-865f-28fdf91ed659
```

Response format

A successful response is a JSON array of node group objects using these keys:

Key	Definition
<code>name</code>	The name of the node group, as 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.

Key	Definition
environment	The name of the node group's environment, as a string. This indirectly defines which classes are available to declare on the node group, and this is the environment that nodes in this node group run in.
environment_trumps	This is a Boolean that changes the response to conflicting environment classifications. By default, if a node belongs to multiple groups with different environments, a <code>classification-conflict</code> error is returned. If the <code>environment_trumps</code> flag is set on a node group, then that node group's environment overrides environments of other groups (if the other groups do not have this flag set), and no attempt is made to validate that the other node groups' classes and class parameters exist in this node group's environment. This is used, for example, with Environment-based testing on page 451.
parent	The ID of the node group's parent, as a string. The only node group without a parent is the All Nodes group, which is the root of the node group hierarchy. The root group, All Nodes , always has the lowest-possible random UUID, which is: 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.
classes	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, which is always a string.
config_data	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, which is always a string. This feature is enabled or disabled through the <code>classifier::allow-config-data</code> setting. When set to <code>false</code> , this key is omitted.

Key	Definition
deleted	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 omitted. Checking for the presence of this key is an easy way to check whether the node group has references that need to be updated. The keys of this object are class names, and the values are also 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> , which is a Boolean indicating whether the specific class parameter has been deleted, and <code>value</code> , which is the string value set by the node group for this parameter (the value is duplicated for convenience; also appears in the <code>classes</code> object).
variables	An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).
last_edited	The most recent time at which a user committed changes to a node group. This is a time stamp in ISO 8601 format, YYYY-MM-DDTHH:MM:SSZ.
serial_number	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.

For example, this response describes a single node group:

```
{
  "environment_trumps": false,
  "parent": "00000000-0000-4000-8000-000000000000",
  "name": "My Nodes",
  "variables": {},
  "id": "085e2797-32f3-4920-9412-8e9decf4ef65",
  "environment": "production",
  "classes": {}
}
```

This example also describes a single node group, but with more information:

```
{
  "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 includes 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": {}
}

```

In the above example, the entire payload class has been deleted, because the `puppetlabs.classifier/deleted` key maps to `true`. This is in contrast to the rocket class, which has had only its `stages` parameter deleted.

POST /v1/groups

Create a node group with a randomly-generated ID.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must be a JSON object describing the node group to be created. The request uses these keys (which are required unless otherwise noted):

Key	Definition
name	The name of the node group, as a string.
environment	The name of the node group's environment. This is optional. If omitted, the default value is <code>production</code> .
environment_trumps	When a node belongs to two or more groups, this Boolean indicates whether this node group's environment overrides environments defined by other node groups. This is optional. If omitted, the default value is <code>false</code> .
description	A string describing the node group. This is optional. If omitted, the node group has no description.
parent	The ID of the node group's parent. This is required.
rule	The condition that must be satisfied for a node to be classified into this node group. For rule formatting assistance, refer to Forming node classifier API requests on page 514.
variables	An optional object that defines the names and values of any top-level variables set by the node group. Supply key-value pairs of variable names and corresponding variable values. Variable values can be any type of JSON value. The <code>variables</code> object can be omitted if the node group does not define any top-level variables.
classes	A required object that defines the classes to be used by nodes in the node group. The <code>classes</code> object contains the parameters for each class. Some classes have required parameters. This object contains nested objects – The <code>classes</code> object's keys are class names (as strings), and each key's value is an object that defines class parameter names and their values. Within the nested objects, the keys are the parameter names (as strings), and each value is the parameter's assigned value (which can be any type of JSON value). If no classes are declared, then <code>classes</code> must be supplied as an empty object ({}). If missing, the server returns a 400 Bad request response.

Key	Definition
config_data	<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. No configuration data is stored if you supply a <code>config_data</code> object that only contains a class name, such as <code>"config_data": {"qux": {}}</code>.</p> <p>Note: This feature is enabled by the <code>classifier::allow-config-data</code> setting. When set to <code>false</code>, supplying the <code>config_data</code> object triggers a 400 response.</p>

For example, this request creates 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"
```

Response format

If the node group was successfully created, the service returns a 303 See Other response with a URI path you can use with the [GET /v1/groups<id>](#) on page 524 endpoint to retrieve data for the new node group.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key. There are several errors you might encounter with the POST `/v1/groups` endpoint:

Response code	Message	Description
400 Bad Request	schema-violation	Required keys are missing or the value of any supplied key does not match the required type.
400 Bad Request	malformed-request	The request's body could not be parsed as JSON.
422 Unprocessable Entity	uniqueness-violation	The request content violates uniqueness constraints. For example, each node group name must be unique within distinct environments. The error response describes the invalid field.

Response code	Message	Description
422 Unprocessable Entity	missing-referents	<p>Classes or class parameters declared on the node group, or inherited by the node group from its parent, do not exist in the specified environment. The error response lists the missing classes or parameters. The <code>details</code> contains an array of objects, where each object uses these keys to describe a single missing referent:</p> <ul style="list-style-type: none"> • <code>kind</code>: Either "missing-class" or "missing-parameter", depending on whether the entire class doesn't exist, or the parameter is missing from the class. • <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. • <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 is actually defined. • <code>defined_by</code>: The name of the node group that defines the class or parameter. This can be the same as <code>group</code> or a parent of <code>group</code>.
422 Unprocessable Entity	missing-parent	The specified parent node group does not exist.

Response code	Message	Description
422 Unprocessable Entity	inheritance-cycle	The request causes an inheritance cycle. The error response contains a description of the cycle, including a list of the node group names, where each node group is followed by its parent until the first node group is repeated.

GET /v1/groups/<id>

Retrieve a specific node group.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. For example:

```
GET https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65
```

Response format

The successful response is the same as the [GET /v1/groups](#) on page 516 endpoint, except that the response describes only a single node group, rather than all node groups.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If the endpoint can't find a node group with the specified ID, the server returns a 404 Not Found or malformed-UUID response.

Related information

[Node classifier API errors](#) on page 562

Learn about node classifier API error responses.

PUT /v1/groups/<id>

Create a node group with a specific ID.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain the ID you want to assign to the new group. The ID must be a valid type-4 (random) UUID. The regular expression used to validate node group UUIDs is `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`.



CAUTION: If you use an ID belonging to an existing node group, that node group is overwritten by the new group.

It is possible to overwrite an existing node group with a new node group definition that contains deleted classes or parameters.

The request body must be a JSON object describing the node group to be created. The request uses these keys (which are required unless otherwise noted):

Key	Definition
name	The name of the node group, as a string.
environment	The name of the node group's environment. This is optional. If omitted, the default value is <code>production</code> .
environment_trumps	When a node belongs to two or more groups, this Boolean indicates whether this node group's environment overrides environments defined by other node groups. This is optional. If omitted, the default value is <code>false</code> .
description	A string describing the node group. This is optional. If omitted, the node group has no description.
parent	The ID of the node group's parent. This is required.
rule	The condition that must be satisfied for a node to be classified into this node group. For rule formatting assistance, refer to Forming node classifier API requests on page 514.
variables	An optional object that defines the names and values of any top-level variables set by the node group. Supply key-value pairs of variable names and corresponding variable values. Variable values can be any type of JSON value. The <code>variables</code> object can be omitted if the node group does not define any top-level variables.
classes	A required object that defines the classes to be used by nodes in the node group. The <code>classes</code> object contains the parameters for each class. Some classes have required parameters. This object contains nested objects – The <code>classes</code> object's keys are class names (as strings), and each key's value is an object that defines class parameter names and their values. Within the nested objects, the keys are the parameter names (as strings), and each value is the parameter's assigned value (which can be any type of JSON value). If no classes are declared, then <code>classes</code> must be supplied as an empty object ({}). If missing, the server returns a 400 Bad request response.

Response format

If the node group is successfully created, the service returns a 201 `Created` response and a JSON body describing the node group.

If the node group already exists, and the existing group is identical to the node group described in the request, then the server takes no action, returns a 200 `OK` response, and a JSON body describing the node group.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key, and are similar to the [POST /v1/groups](#) on page 521 error responses. However, 422 responses to POST requests can

include errors caused by a node group's children, but a node group being created with a PUT request cannot have any children.

POST /v1/groups/<id>

Edit the name, environment, parent node group, rules, classes, class parameters, configuration data, and variables for a specific node group.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must be a JSON object describing the changes you want to make to the node group. For a complete list of keys you can include in the request, refer to [POST /v1/groups](#) on page 521.

If your request includes `classes`, `config_data`, `variables`, and `rule` keys, these values are merged with the node group's existing values. Any keys in the resulting combined object with a null value are removed. To remove classes, class parameters, configuration data, variables, or rules from the node group, set the key to null in the change request.

If the request supplies a `rule` key that is set to a new value or `nil`, the rule is updated wholesale or removed from the group, depending on the supplied value.

If the request contains the `name`, `environment`, `description`, or `parent` keys, then these values replace the old values entirely.

The `serial_number` key is optional. If your request includes a `serial_number` that does not match the group's current serial number, the service returns a `409 Conflict` response. To bypass this check, omit the `serial_number` key from the request.

The following code examples show a node group, the change request, and the end result (where the changes are merged into the node group's settings).

The original node group settings:

```
{
  "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"]
  }
}
```

The change request:

```
{
  "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" ]
}
}

```

The updated group settings:

```

{
  "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" ]
  }
}

```

In the above example, 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.

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.

Here is an example of a complete curl request to the `POST /v1/groups/<id>` endpoint:

```

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"

```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key and are similar to the [POST /v1/groups](#) on page 521 error responses.

You can't edit the root **All Nodes** node group's `rule`. Attempting to do so returns a `422 Unprocessable Entity` response.

DELETE /v1/groups/<id>

Use the `/v1/groups/\<id\>` endpoint to delete the node group with the given ID.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. For example:

```
DELETE https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65
```

You can use [GET /v1/groups](#) on page 516 to get node group IDs.

Response format

If the node group is successfully deleted, the sever returns a `204 No Content` response.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

If the endpoint can't find a node group with the specified ID, the server returns a `404 Not Found` or `malformed-UUID` response.

You can't delete node groups that have children. If you attempt to delete a node group with children, the server returns a `422 Unprocessable Entity children-present` response. To resolve the error, you must either delete the children or reassign the children to a new parent (such as with the [POST /v1/groups/<id>](#) on page 526 endpoint).

Related information

[Node classifier API errors](#) on page 562

Learn about node classifier API error responses.

POST /v1/groups/<id>/pin

Pin specific nodes to a node group.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. For example:

```
POST https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin
```

The request must also provide the names of the nodes you want to pin to the group. There are two ways to do this:

- Append the node names to the URI path. If you are pinning more than one node, use encoded comma separation (%2C) between node names. For example, this request pins the nodes named `foo`, `bar`, and `baz` to the group:

```
POST https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/pin?nodes=foo%2Cbar%2Cbaz
```

- Supply the node names in the request body. For a single node, you can supply this in a simple JSON object. For multiple nodes, supply the node names in an array. For example, this JSON body pins a single node:

```
{ "nodes": [ "foo" ] }
```

And this body pins three nodes:

```
{ "nodes": [ "foo", "bar", "baz" ] }
```

While it's easier to append the nodes to the end of the URI path, if you want to pin a lot of nodes at once, the URI path might get truncated. Strings are truncated if they exceed 8,000 characters. In this case, you have to supply the nodes in a JSON body, which can be many megabytes in size.

Here is an example of a complete curl request to the `POST /v1/groups/<id>/pin` endpoint. If necessary, replace `/etc/puppetlabs/puppet/ssl/certs/ca.pem` with the correct path to your CA certificate file:

```
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 --cacert "/etc/puppetlabs/puppet/ssl/certs/ca.pem" --header
"$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

Response format

If pinning is successful, the service returns a `204 No Content` response with an empty body.

If the request contained a node that is already pinned to the group, the node's pinned status is unchanged — The service only pins nodes that aren't already pinned to the specified node group.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

If your request doesn't specify any nodes to pin, the service returns a `400 Malformed Request` response.

If the request body is invalid JSON, is missing the `nodes` key, or contains any keys other than `nodes`, the service returns a `400 Malformed Request` response.

POST /v1/groups/<id>/unpin

Unpin specific nodes from a node group.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. For example:

```
POST https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/unpin
```

The request must also provide the names of the nodes you want to unpin from the group. There are two ways to do this:

- Append the node names to the URI path. If you are unpinning more than one node, use encoded comma separation (%2C) between node names. For example, this request unpins the nodes named `foo`, `bar`, and `baz` from the group:

```
POST https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/unpin?nodes=foo%2Cbar%2Cbaz
```

- Supply the node names in the request body. For a single node, you can supply this in a simple JSON object. For multiple nodes, supply the node names in an array. For example, this JSON body unpins a single node:

```
{ "nodes": [ "foo" ] }
```

And this body unpins three nodes:

```
{ "nodes": [ "foo", "bar", "baz" ] }
```

While it's easier to append the nodes to the end of the URI path, if you want to unpin a lot of nodes at once, the URI path might get truncated. Strings are truncated if they exceed 8,000 characters. In this case, you have to supply the nodes in a JSON body, which can be many megabytes in size.

Here is an example of a complete curl request to the POST /v1/groups/<id>/pin endpoint:

```
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/unpin"
data='{ "nodes": [ "example-to-unpin" ] }'

curl --header "$type_header" --header "$auth_header" --request POST "$uri"
--data "$data"
```

Tip: You can use the [POST /v1/commands/unpin-from-all](#) on page 543 endpoint to unpin specific nodes from all groups they're pinned to.

Response format

If unpinning is successful, the service returns a 204 No Content response with an empty body.

If the request contained a node that is was not pinned to the group, service ignores that node.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If your request doesn't specify any nodes to unpin, the service returns a 400 Malformed Request response.

If the request body is invalid JSON, is missing the nodes key, or contains any keys other than nodes, the service returns a 400 Malformed Request response.

GET /v1/groups/<id>/rules

Resolve the rules for a specific node group, and then translate those rules to work with the PuppetDB nodes and inventory endpoints.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. For example:

```
GET https://localhost:4433/classifier-api/v1/
groups/085e2797-32f3-4920-9412-8e9decf4ef65/rules
```

Response format

A successful response returns a JSON object that uses these keys to enumerate the group's rules:

Key	Definition
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 (nodes_query_format and inventory_query_format), which represent 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.

For example:

```
{
  "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

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

If the endpoint can't find a node group with the specified ID, the server returns a `404 Not Found` or `malformed-UUID` response.

Related information

[Error response description](#) on page 562

Node classifier API error responses are formatted as JSON objects.

Classes endpoint

Use the `classes` endpoint to retrieve a list of all classes.

To retrieve a list of all classes in a specific environment or a specific class from a specific environment, use the [GET /v1/environments/<environment>/classes](#) on page 546 or [GET /v1/environments/<environment>/classes/<name>](#) on page 547 endpoints.

The output from the `classes` endpoint (and the `environments/<environment>/classes` endpoints) is useful for creating or editing node groups, which usually reference one or more classes. You can use the [Groups endpoints](#) on page 516 to create and edit node groups.

The node classifier gets information about classes from Puppet. Don't use the `classes` endpoint to create, update, or delete classes.

GET /v1/classes

Retrieve a list of all classes the node classifier knows about at the time of the request.

Request format

The `GET /v1/classes` endpoint returns the node classifier's current class data. The node classifier periodically retrieves class data from the primary server, and you can check the last retrieval time with the [GET /v1/last-class-update](#) on page 558 endpoint. If you want to ensure the response contains the latest data, use the [POST /v1/update-classes](#) on page 558 endpoint to force a retrieval.

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication.

Response format

A successful response is a JSON array of objects. Each object uses these keys to describe a class:

Key	Definition
name	The name of the class, as 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.

This is an example of one class object:

```
{
  "name": "apache",
  "environment": "production",
  "parameters": {
    "default_mods": true,
    "default_vhost": true,
    ...
  }
}
```

Classification endpoints

The classification endpoints accept a node name and a set of facts, and then return information about how the specified node is classified. The output can help you test your node group classification rules.

POST /v1/classified/nodes/<name>

Retrieve a specific node's classification information based on facts supplied in the body of your request.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain the name of a specific node, and the body can contain a JSON object using these keys:

Key	Definition
fact	A JSON object containing regular, non-trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.
trusted	A JSON object containing trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.

Tip: There is also a [POST /v2/classified/nodes/<name>](#) on page 563 endpoint.

Here is an example of a curl command for the /v1/classified/nodes/<name> endpoint:

```
type_header='Content-Type: application/json'
```

```

auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v1/
classified/nodes/<NAME>"
data='{"fact" : { "<FACT_NAME>" : "<FACT_VALUE>" }}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"

```

Response format

A successful response returns a JSON object using these keys to describe the node's classification:

Key	Definition
name	The node name, as a string.
groups	An array of group IDs for the groups that the node was classified into.
environment	The name of the environment that the node uses, which is taken from the node groups the node was classified into.
classes	An object containing key/value pairs describing the classes that this node received from the groups it was classified into.
parameters	Each key/value pair consists of a class name, as a string, and a subsequent object containing the names and values of parameters within the named class.
parameters	An object containing key/value pairs of top-level variables and their assigned values.

For example:

```
{
  "name": "foo.example.com",
  "groups": ["9c0c7d07-a199-48b7-9999-3cdf7654e0bf", "96d1a058-225d-48e2-
a1a8-80819d31751d"],
  "environment": "staging",
  "parameters": {},
  "classes": {
    "apache": {
      "keepalive_timeout": 30,
      "log_level": "notice"
    }
  }
}
```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

If the node is classified into multiple node groups that supply conflicting classifications to the node, the server returns a 500 Server error response.

For classification-conflict errors, the `msg` describes generally why the conflict happened, and the `details` contains an object that uses the `environment`, `variables`, or `classes` key to indicate the type of

conflict (whether it was in setting the environment, setting variables, or setting class parameters). Each key contains value-detail objects describing the specific conflicts:

Value-detail object key	Definition
value	The specific value having a conflict. For environment and classes, these are strings. For variables, these can be any JSON value type.
from	The node group that the node was classified into that caused the conflicting value to be added to the node's classification. Refer to <code>defined_by</code> for further details.
defined_by	The node group that actually defined the conflicting value. This is often the <code>from</code> group, but could be an ancestor of that group, due to How node group inheritance works on page 440.

The following example demonstrates a conflicting value being inherited from an ancestor group and a conflicting value supplied directly from the assigned node group. The conflicting value Blue Suede Shoes was included in the classification because the node matched the Elvis Presley group (as indicated by `from`). However, the conflicting value was actually defined by the Carl Perkins group, which is an ancestor of the Elvis Presley group. This caused the child group to inherit the value from the ancestor group. The Since You've Been Gone conflicting value is defined by the same group that the node was assigned to.

```
{
  "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"}},
              ...
            }
          }
        ]
      }
    }
  }
}
```

```

        ]
    }
}
}
```

Related information

[Node classifier API errors](#) on page 562

Learn about node classifier API error responses.

POST /v1/classified/nodes/<name>/explanation

Retrieve a detailed explanation about how a node is classified based on facts supplied in the body of your request.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain the name of a specific node, and the body can contain a JSON object using these keys:

Key	Definition
fact	A JSON object containing regular, non-trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.
trusted	A JSON object containing trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.

For example:

```
{
  "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 describing how the node would be classified based on the submitted facts.

- If the node would be successfully classified, the response object contains the successful classification outcome.
- If the classification would fail due to conflicts, the response object describes the conflicts.

The response is intended to provide insight into the classification process, so that, if a node isn't classified as expected, you can trace the classification sequence to the source of the deviation.

Classification proceeds in this order:

1. All node group rules are tested on the node's facts and name. Groups that don't match the node are culled, leaving only the matching groups. This step contributes to the `match_explanations` key in the response body.

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. The remaining node groups are referred to as *leaf groups*. This step contributes to the `leaf_groups` key in the response body.
3. Each leaf group is transformed into its inherited classification by adding all the inherited class and class parameter values from their ancestors. This step contributes to the `inherited_classifications` key in the response body.
4. All inherited classifications and individual node classifications are inspected for conflicts. A conflict occurs whenever two inherited classifications define different values for the same environment, class parameter, or top-level variable. This step contributes to the `conflicts` key in the response body.
5. Any individual node classifications (including classes, class parameters, configuration data, and variables) are added. This step contributes to the `individual_classification` key in the response body.
6. Individual node classifications are applied to the group classification, which forms the final classification. This step contributes to the `final_classification` key in the response body.

The response's JSON object uses these keys to describe the classification:

Key	Definition
<code>match_explanations</code>	An object containing group ID's the node matched to. For each group ID, there is an object explaining why the node matched that particular group's rules.
<code>leaf_groups</code>	This key's value is an array of the leaf groups. This represent a condensed list of matching groups after filtering out any matching node group that had a descendant that was also a matching node group.
<code>inherited_classifications</code>	This key's value is an object mapping a leaf group's ID to the classification values provided by that group (including inheritance from ancestors).
<code>conflicts</code>	This key is present only if there are conflicts in the inherited classifications. For each conflict there is an array of conflict details. 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 classification provided the conflicting 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>	This key's value includes classes, class parameters, configuration data, and variables applied directly to the node.
<code>final_classification</code>	This key is present only if there are no conflicts between the inherited classifications. Its value is the result of merging all individual node classifications and group classifications.
<code>node_as_received</code>	Represents the node object as defined in the request, including the name and facts (if supplied).
<code>classification_source</code>	An annotated version of the node's classification where the environment, each class parameter, and each variable are replaced with an annotated value object.

This example shows a successful (non-conflicting) classification response:

```
{
```

```

"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" } ],
        {
          "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": [
                {
                  "path": [ "fact", "eyebrow pitch" ],
                  "value": "30"
                }
              ]
            }
          ]
        }
      }
    ]
  }
},
"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",
                "order": 2
            }
        }
    }
}
}

```

```
        "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 conflicting classification response:

```
{  
  "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"
  }
}
}

```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

Commands endpoint

Use the `commands` endpoint to unpin specified nodes from all node groups they're pinned to.

If you want to unpin one or more nodes from a single node group, use the [POST /v1/groups/<id>/unpin](#) on page 529 endpoint.

To re-pin nodes you've unpinned, use the [POST /v1/groups/<id>/pin](#) on page 528 endpoint.

POST /v1/commands/unpin-from-all

Unpin one or more specific nodes from all node groups they're pinned to. Unpinning has no effect on nodes that are assigned to node groups via dynamic rules.

Request format

If you submit a request to the `/v1/commands/unpin-from-all` endpoint, the endpoint only removes the specified nodes from groups that you have permission to view and edit. Because group permissions are applied hierarchically, you must have one of the following permissions for the **parent** groups of each group you want to unpin nodes from:

- **Create, edit, and delete child groups**
- **Edit child group rules**

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must be a JSON object containing the certnames of the nodes you want to unpin. For a single node, you can supply this in a simple JSON object. For multiple nodes, supply the certnames in an array. For example, this JSON body unpins a single node:

```
{ "nodes": "foo" }
```

And this body unpins three nodes:

```
{ "nodes": [ "foo", "bar", "baz" ] }
```

Here is an example of a complete curl request for this endpoint:

```
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"
```

Tip: Nodes assigned to groups via dynamic rules can't be unpinned. If you want to remove dynamically-classified nodes from groups, you need to modify the group's rules. To explore a node's classification, use the [Classification endpoints](#) on page 533.

Response format

Note: If you have a lot of node groups, the endpoint takes longer to respond.

If unpinning is successful, the service returns a list of nodes and the groups they were unpinned from. If a node you specified in your request was not pinned to any groups, that node is omitted from the response. Here are two response examples:

```
{ "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" } ] } ] }
```

```
{ "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" } ] } ] }
```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If your request doesn't specify any nodes to unpin, the service returns a 400 Malformed Request response.

If the request body is invalid JSON, is missing the nodes key, or contains any keys other than nodes, the service returns a 400 Malformed Request response.

Environments endpoints

Use the environments endpoints to retrieve the node classifier's environment data. The responses tell you which environments are available, whether a named environment exists, and which classes exist in a certain environment.

The responses are useful for creating node groups, which must be associated with an environment. You can use the [Groups endpoints](#) on page 516 to create and edit node groups.

The node classifier gets environment information from Puppet. Do not use the environments endpoints to create, update, or delete environments.

GET /v1/environments

Retrieve a list of all environments the node classifier knows about at the time of the request.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication.

Response format

A successful response is a JSON array of objects. Each object uses these keys to describe an environment:

Key	Definition
name	The name of the environment, as a string.
sync_succeeded	A Boolean indicating whether the environment synced successfully during the last class synchronization.

GET /v1/environments/<name>

Retrieve information about a specific environment. This endpoint is useful for checking whether an environment exists.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication. The URI path must specify an environment name, such as production. For example:

```
GET https://localhost:4433/classifier-api/v1/environments/production
```

Response format

If the environment exists, the endpoint returns a 200 response and a JSON array containing one object. The object uses these keys to describe the specified environment:

Key	Definition
name	The name of the environment, as a string.
sync_succeeded	A Boolean indicating whether the environment synced successfully during the last class synchronization.

Error responses

If there is no environment with the specified name, the endpoint returns a 404 Not Found response with an empty body. Other possible errors follow the usual format for [Node classifier API errors](#) on page 562.

PUT /v1/environments/<name>

Create a new environment with a specific name.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain the name you want to assign to the new environment. For example, this request creates an environment called `staging`:

```
PUT https://localhost:4433/classifier-api/v1/environments/staging
```

Response format

If the environment is successfully created, the service returns a `201 Created` response and a JSON body describing the environment.

If an environment with the given name already exists, the endpoint might return a `200 OK` response.

Error responses

Possible errors follow the usual format for [Node classifier API errors](#) on page 562.

GET /v1/environments/<environment>/classes

Retrieve a list of all classes (that the node classifier knows about) in a specific environment.

Request format

The `/v1/environments/<environment>/classes` endpoint returns the node classifier's current class data for the specified environment. The node classifier periodically retrieves class data from the primary server, and you can check the last retrieval time with the [GET /v1/last-class-update](#) on page 558 endpoint. If you want to ensure the response contains the latest data, use the [POST /v1/update-classes](#) on page 558 endpoint to force a retrieval. To get a list of all classes for all environments, use the [GET /v1/classes](#) on page 532 endpoint.

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication. The URI path must specify an environment, such as `production`. For example:

```
GET https://localhost:4433/classifier-api/v1/environments/production/classes
```

You can use the [GET /v1/environments](#) on page 545 endpoint to get a list of known environments.

Response format

A successful response is a JSON array of objects. Each object uses these keys to describe a class:

Key	Definition
<code>name</code>	The name of the class, as 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.

This is an example of one class object:

```
{
  "name": "apache",
  "environment": "production",
  "parameters": {
    "default_mods": true,
    "default_vhost": true,
    ...
  }
}
```

For errors, refer to [Node classifier API errors](#) on page 562.

GET /v1/environments/<environment>/classes/<name>

Retrieve the class with the given name in the given environment.

Request format

The /v1/environments/<environment>/classes/<name> endpoint returns the node classifier's current class data for the specified environment and class. The node classifier periodically retrieves class data from the primary server, and you can check the last retrieval time with the [GET /v1/last-class-update](#) on page 558 endpoint. If you want to ensure the response contains the latest data, use the [POST /v1/update-classes](#) on page 558 endpoint to force a retrieval. To get a list of all classes in an environment, use the [GET /v1/environments/<environment>/classes](#) on page 546 endpoint. To get a list of all classes in all environments, use the [GET /v1/classes](#) on page 532 endpoint.

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication. The URI path must specify an environment, such as `production`, and a class name. For example:

```
GET https://localhost:4433/classifier-api/v1/environments/production/
classes/apache
```

Response format

A successful response is a JSON array containing one object that uses these keys to describe the class:

Key	Definition
<code>name</code>	The name of the class, as 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 null. If the value is null, the parameter is required.

For example:

```
{
  "name": "apache",
  "environment": "production",
  "parameters": {
```

```

    "default_mods": true,
    "default_vhost": true,
    ...
}
}

```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

If the endpoint can't find a class with the specified name, the server returns a `404 Not Found` response with an empty body.

Nodes check-in history endpoints

Use the `nodes` endpoints to retrieve records about nodes that have checked into the node classifier.

Enable check-in storage to use this endpoint

By default, node check-in storage is disabled because it can place excessive loads on larger deployments. You must enable node check-in storage to get any information from the `nodes` endpoints. If node check-in storage is disabled, the `nodes` endpoints return empty arrays.

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 446

Configuration data set in the PE console is used for automatic parameter lookup in the same way that Hiera data is used. Console configuration data takes precedence over Hiera data, but you can combine data from both sources to configure nodes.

GET /v1/nodes

Retrieve check-in history for all nodes that have checked in with the node classifier.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication. For example:

```
GET https://localhost:4433/classifier-api/v1/nodes
```

You can append these optional parameters to the URI path:

- `limit`: Set the maximum number of nodes to include in the response. For example, `limit=10` limits the response to 10 nodes. The point at which the `limit` count starts is determined by `offset`.

Tip: The amount of time it takes the endpoint to respond depends on the number of records it has to collect. In deployments that have a very large amount of nodes, the `console-services` process might run out of memory and crash. If your deployment has a lot of nodes, setting the `limit` parameter can expedite the response time and avoid crashes.

- `offset`: Specify a zero-indexed integer at which to start returning results. For example, if you set this to 12, the response returns nodes starting with the 13th record. The default is 0.

For example, this request includes the `limit` and `offset` parameters:

```
GET https://localhost:4433/classifier-api/v1/nodes?limit=25&offset=25
```

Response format

A successful response returns a JSON array of objects. Each object contains:

- `name`: The name of the node, as a string.
- `check_ins`: An array of objects where each object represents a node classifier check-in event.

Each check-in object uses these keys:

- `time`: The check-in time, as a string in ISO-8601 format with timezone.
- `explanation`: An object containing IDs of node groups the node was classified into (during the check-in event) and sub-objects explaining which rule(s) the node matched to be classified into the group. Rule explanation objects use these keys:
 - `value`: A Boolean indicating the result of evaluating the form. At the top level, this is the result of the entire rule condition. Within complex rules, you can use the `value` to trace individual rule condition results. For example, you can check which parts of an `or` condition were true.
 - `form`: A representation of a rule condition. Additional conditions within a complex rule are represented as nested rule explanation objects.

Tip: If you want more detailed classification explanations, use the [POST /v1/classified/nodes/<name>/explanation](#) on page 536 endpoint.

- `transaction_uuid`: A UUID representing a specific Puppet transaction that is submitted by Puppet at the time of the check-in event. This makes it possible to identify the check-in event that generated a specific catalog and report.

Besides the rule condition markup, the comparison operations in the rule conditions 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.

Here is an example of a response object for one node:

```
{
  "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"
    }
  ]
}
```

```
}
```

To help explain the response, assume a node named Deep Space 9 checked into the classifier, and, at check-in time, the node had these facts:

```
"fact": {
  "pressure hulls": "10",
  "docking ports": "18",
  "docking pylons": "3",
  "warp cores": "0",
  "bars": "1"
}
```

Also assume this rule existed for a node group:

```
[ "and", [ ">=", [ "fact", "pressure hulls"], "1"],
  [ "=", [ "fact", "warp cores"], "0"],
  [ ">=", [ "fact", "docking ports"], "10"]]
```

Tip: Refer to [Forming node classifier API requests](#) on page 514 for an explanation of rule condition grammar.

When the Deep Space 9 node checks in for classification, the node's facts caused it to match the rule. When you check the check-in history, the rule explanation object demonstrates the logic behind the rule evaluation that ultimately classified the node into that particular node group. For example:

```
{
  "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"]
    }
  ]
}
```

Error responses

Possible errors follow the usual format for [Node classifier API errors](#) on page 562.

GET /v1/nodes/<node>

Retrieve the check-in history for a specific node.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication. The URI path must specify a node name. For example:

```
GET https://localhost:4433/classifier-api/v1/nodes/Node1234
```

Response format

The response is the same as the [GET /v1/nodes](#) on page 548 endpoint response, but the response only contains information for the specified node.

Error responses

If there is no check-in information for the node with the given name, the endpoint returns a 404 Not Found response. Other possible errors follow the usual format for [Node classifier API errors](#) on page 562.

Group children endpoint

Use the group-children endpoint to retrieve a list of node groups descending from a specific node group.

GET /v1/group-children/<id>

Retrieve a list of node groups descending from a specific node group,

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain a node group ID. The ID must be a valid type-4 (random) UUID. You can use the [GET /v1/groups](#) on page 516 endpoint to retrieve node group IDs.

You can append the optional `depth` parameter to limit how many levels of descendants are returned. For example, `depth=2` limits the response to the group's immediate children and first grandchildren. For example:

```
GET https://localhost:4433/classifier-api/v1/group-children/085e2797-32f3-4920-9412-8e9decf4ef65?depth=2
```

`depth` must be an integer. If `depth=0` the response only returns the base group and no children or grandchildren.

Response format

A successful response returns a JSON array of objects, where each object represents a node group. If grandchildren are present, the objects are nested to represent the node group ancestry tree.

Restriction: The response only contains information about node groups you have permission to view and the children of those node groups. If you specified a `depth`, the response is further constrained. Keep in mind that you might have permission to view a grandchild group but not that group's parent.

- If you only have permission to view the specified group, the response contains the group's descendants.
- If you have permission to view the descendants of the specified group, but not the specified group itself, the response returns children from each ancestry tree you have permission to view.
- If you **do not** have permission to view either the specified group or its descendants, the response is an empty array.

Each node group object uses these keys:

Key	Definition
<code>name</code>	The name of the node group, as 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.

Key	Definition
environment	The name of the node group's environment, as a string. This indirectly defines which classes are available to declare on the node group, and this is the environment that nodes in this node group run in.
environment_trumps	This is a Boolean that changes the response to conflicting environment classifications. By default, if a node belongs to multiple groups with different environments, a <code>classification-conflict</code> error is returned. If the <code>environment_trumps</code> flag is set on a node group, then that node group's environment overrides environments of other groups (if the other groups do not have this flag set), and no attempt is made to validate that the other node groups' classes and class parameters exist in this node group's environment. This is used, for example, with Environment-based testing on page 451.
parent	The ID of the node group's parent, as a string. The only node group without a parent is the All Nodes group, which is the root of the node group hierarchy. The root group, All Nodes , always has the lowest-possible random UUID, which is: 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.
classes	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, which is always a string.

Key	Definition
deleted	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 omitted. Checking for the presence of this key is an easy way to check whether the node group has references that need to be updated. The keys of this object are class names, and the values are also 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> , which is a Boolean indicating whether the specific class parameter has been deleted, and <code>value</code> , which is the string value set by the node group for this parameter (the value is duplicated for convenience; also appears in the <code>classes</code> object).
variables	An object that defines the values of any top-level variables set by the node group. The object is a mapping between variable names and their values (which can be any JSON value).
children	A JSON array containing node group objects for the group's immediate children. Grandchildren are represented in additional nested <code>children</code> arrays. If you included a <code>depth</code> in your request, the amount of nesting stops at the 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 example is a response to a request for the first two levels of children under the root group, `All Nodes`. The user has permission to view only `child-1` and `grandchild-5`, which limits the response to `child-1`, the first children of `child-1`, and `grandchild-5`. No additional grandchildren are represented because the request specified `depth=2`.

```
[  
  {  
    "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-a143-31b278cf859",
        "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
}
]

```

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If the supplied node group ID is not a valid UUID, the server returns a 400 Bad Request malformed-UUID response.

If the value of depth is not an integer, or it is a negative integer, the server returns a 400 Bad Request malformed-number or 400 Bad Request illegal-count response.

If the endpoint can't find a node group with the specified ID, the server returns a 400 Not Found response.

Rules endpoint

Use the `rules` endpoint to translate a node group rule condition into PuppetDB query syntax.

POST /v1/rules/translate

Translate a node group rule condition into PuppetDB query syntax.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must be a JSON object describing a rule condition. The rule condition must be structured like a `rule` key for a node group object. To get examples of `rule` keys, use the [GET /v1/groups/<id>/rules](#) on page 530 endpoint.

You can append the optional `format` parameter to the end of the URI path to change the response format. The default value is `nodes`. If you specify `format=inventory`, the response returns classifier rules in a compatible [dot notation](#) format, instead of the [PuppetDB AST](#) format.

Response format

If you did not specify the `format` in your request, or if you specified `format=nodes`, then the response is a PuppetDB query string. You can use this query string with `nodes` endpoint in PuppetDB to get a list of nodes matching the rule condition.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key.

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 can't be translated into a PuppetDB query, the server returns a `422 Unprocessable Entity untranslatable-rule` response with a message describing why the rule can't be translated and a copy of your supplied rule.

If the request doesn't contain a valid rule, the server returns a `400 Bad Request` response. There are two common variations of this error:

- `malformed-request`: The rule is not valid JSON. The error response body contains a copy of your supplied rule.
- `schema-violation`: The rule is valid JSON but the rule grammar is incorrect. Refer to [Forming node classifier API requests](#) on page 514 for information about rule grammar. The `details` key in the error response body describes the submitted rule object, the schema the object was expected to conform to, and how the submitted object failed to conform to the schema.

Related information

[GET /v1/groups](#) on page 516

Retrieves 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 defined in the body of the request.

POST /v1/import-hierarchy

Delete *all* existing node groups from the node classifier service and replace them with the node groups defined in the body of the submitted request.

Request format



CAUTION: This endpoint deletes **all** existing node groups, and then creates new node groups based on the content of the request body.

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must contain an array of node group objects that form a valid and complete node group hierarchy. *Valid* means that the hierarchy does not contain any cycles (self-referencing inheritance loops). *Complete* means that every node group in the hierarchy is reachable from the root node group (**All Nodes**).

Tip: Responses from the [GET /v1/groups](#) on page 516 endpoint are valid input for the `/v1/import-hierarchy` endpoint.

The request body must be a JSON array containing JSON objects describing the node groups to be created. Use the following keys. You must specify **all** keys – This endpoint supplies no default values.

Key	Definition
<code>name</code>	The name of the node group, as a string.
<code>environment</code>	The name of the node group's environment, such as <code>production</code> .
<code>environment_trumps</code>	When a node belongs to two or more groups, this Boolean indicates whether this node group's environment overrides environments defined by other node groups.
<code>description</code>	A string describing the node group. For no description, supply an empty string.
<code>parent</code>	The ID of the node group's parent.
<code>rule</code>	The condition that must be satisfied for a node to be classified into this node group.
<code>variables</code>	For rule formatting assistance, refer to Forming node classifier API requests on page 514.
<code>variables</code>	An object that defines the names and values of any top-level variables set by the node group. Supply key-value pairs of variable names and corresponding variable values. Variable values can be any type of JSON value. The <code>variables</code> object can be empty if the node group does not define any top-level variables.
<code>classes</code>	A object that defines the classes to be used by nodes in the node group. The <code>classes</code> object contains the parameters for each class. Some classes have required parameters. This object contains nested objects – The <code>classes</code> object's keys are class names (as strings), and each key's value is an object that defines class parameter names and their values. Within the nested objects, the keys are the parameter names (as strings), and each value is the parameter's assigned value (which can be any type of JSON value). If no classes are declared, then <code>classes</code> must be supplied as an empty object (<code>{}</code>).

Key	Definition
config_data	<p>An 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. No configuration data is stored if you supply a <code>config_data</code> object that only contains a class name, such as <code>"config_data": {"qux": {}}</code>.</p> <p>Note: This key is enabled by the <code>classifier::allow-config-data</code> setting. When set to <code>false</code>, supplying the <code>config_data</code> object triggers a 400 response.</p>

Response format

If the submitted request forms a complete and valid node group hierarchy, and the replacement operation is successful, the endpoint returns a 204 No Content response with an empty body.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key. There are several errors you might encounter with the POST /v1/import-hierarchy endpoint:

Response code	Message	Description
400 Bad Request	schema-violation	Keys are missing or the value of any supplied key does not match the required type.
400 Bad Request	malformed-request	The request's body could not be parsed as JSON.
422 Unprocessable Entity	unreachable-groups	The node group hierarchy contains node groups that are unreachable from the root node group. The response lists the unreachable node groups.
422 Unprocessable Entity	inheritance-cycle	The request causes an inheritance cycle. The error response contains a description of the cycle, including a list of the node group names, where each node group is followed by its parent until the first node group is repeated.

You might also encounter some of the errors returned by the [POST /v1/groups](#) on page 521 endpoint.

Related information

[Groups endpoints](#) on page 516

The `groups` endpoints create, read, update, and delete groups.

[Node classifier API errors](#) on page 562

Learn about node classifier API error responses.

Last class update endpoint

Use the `last-class-update` endpoint to retrieve the time that classes were last updated from the primary server.

To prompt the node classifier to fetch updated class and environment definitions from the primary server, use the [Update classes endpoint](#) on page 558.

GET /v1/last-class-update

Retrieve the time that classes were last updated from the primary server.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the request is a basic GET call with authentication.

Response format

A successful response is a JSON object containing `last_update`. If there has been an update, `last_update` reports the time of the last update in ISO-8601 format. If the node classifier has never updated the classes from the primary server, `last_update` is null.

Update classes endpoint

Use the `update-classes` endpoint to trigger the node classifier to get updated class and environment definitions from the primary server.

Restriction: If you changed the value of the `environment_class_cache_enabled` setting on your primary server to `true` **and** you **don't** use Code Manager, you must [manually delete the environment cache](#) before using the `update-classes` endpoint.

To check the last time the node classifier got updated definitions from the primary server, use the [Last class update endpoint](#) on page 558.

Related information

[Use cached data when updating classes](#) on page 216

The `environment_class_cache_enabled` setting specifies whether cached data is used when updating classes in the Puppet Enterprise (PE) console. When `true`, Puppet Server uses file sync when refreshing classes, which provides improved performance.

POST /v1/update-classes

Trigger the node classifier to retrieve updated class and environment definitions from the primary server. The classifier service also uses this endpoint when you refresh classes in the console.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the default request updates definitions for all environments. If you only want to update a specific environment, append the `environment` parameter to the URI path. For example, this request only updates definitions for the production environment:

```
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"
```

Response format

If the definitions were successfully updated, the service returns a 201 response with an empty body.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If the node classifier gets an unexpected status from the primary server, the service returns 500 Server Error unexpected-response and a copy of the response from the primary server.

Related information

[Use cached data when updating classes](#) on page 216

The environment_class_cache_enabled setting specifies whether cached data is used when updating classes in the Puppet Enterprise (PE) console. When true, Puppet Server uses file sync when refreshing classes, which provides improved performance.

Validation endpoint

Use the validation endpoint to validate groups in the node classifier.

POST /v1/validate/group

Validate groups in the node classifier.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the body must be a JSON object describing the node group to be validated. The request uses these keys (which are required unless otherwise noted):

Key	Definition
name	The name of the node group, as a string.
environment	The name of the node group's environment. This is optional. If omitted, the default value is production.
environment_trumps	When a node belongs to two or more groups, this Boolean indicates whether this node group's environment overrides environments defined by other node groups. This is optional. If omitted, the default value is false.
description	A string describing the node group. This is optional. If omitted, the node group has no description.
parent	The ID of the node group's parent. This is required.
rule	The condition that must be satisfied for a node to be classified into this node group.
variables	For rule formatting assistance, refer to Forming node classifier API requests on page 514.
	An optional object that defines the names and values of any top-level variables set by the node group. Supply key-value pairs of variable names and corresponding variable values. Variable values can be any type of JSON value. The variables object can be omitted if the node group does not define any top-level variables.

Key	Definition
classes	A required object that defines the classes to be used by nodes in the node group. The <code>classes</code> object contains the parameters for each class. Some classes have required parameters. This object contains nested objects – The <code>classes</code> object's keys are class names (as strings), and each key's value is an object that defines class parameter names and their values. Within the nested objects, the keys are the parameter names (as strings), and each value is the parameter's assigned value (which can be any type of JSON value). If no classes are declared, then <code>classes</code> must be supplied as an empty object (<code>{ }</code>). If missing, the server returns a <code>400 Bad request</code> response.

For example, this request validates 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/validate/
group"
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"
```

Response format

If the group is valid, the service returns a `200 OK` response with the validated group's information in the body.

Error responses

If there is an error, [Node classifier API errors](#) on page 562 provide error information in the `kind` key. There are several errors you might encounter with the `/v1/validate/group` endpoint:

Response code	Message	Description
400 Bad Request	schema-violation	Required keys are missing or the value of any supplied key does not match the required type.
400 Bad Request	malformed-request	The request's body could not be parsed as JSON.
422 Unprocessable Entity	uniqueness-violation	The request content violates uniqueness constraints. For example, each node group name must be unique within distinct environments. The error response describes the invalid field.

Response code	Message	Description
422 Unprocessable Entity	missing-referents	<p>Classes or class parameters declared on the node group, or inherited by the node group from its parent, do not exist in the specified environment. The error response lists the missing classes or parameters. The <code>details</code> contains an array of objects, where each object uses these keys to describe a single missing referent:</p> <ul style="list-style-type: none"> • <code>kind</code>: Either "missing-class" or "missing-parameter", depending on whether the entire class doesn't exist, or the parameter is missing from the class. • <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. • <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 is actually defined. • <code>defined_by</code>: The name of the node group that defines the class or parameter. This can be the same as <code>group</code> or a parent of <code>group</code>.
422 Unprocessable Entity	missing-parent	The specified parent node group does not exist.

Response code	Message	Description
422 Unprocessable Entity	inheritance-cycle	The request causes an inheritance cycle. The error response contains a description of the cycle, including a list of the node group names, where each node group is followed by its parent until the first node group is repeated.

Node classifier API errors

Learn about node classifier API error responses.

Error response description

Node classifier API error responses are formatted as JSON objects.

Error responses use these keys:

Key	Definition
kind	The kind of error encountered.
msg	The message associated with the error.,
details	A hash containing additional information about the error. This information might be less user-friendly than the msg.

Internal server errors

Endpoints might return 500: Internal Server Error responses in addition to their 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 500 application-error response contains the underlying error's message and a description of information contained in details. The details contain the error's stack trace (as an array of strings) and might contain schema, value, and error keys if the error was caused by a schema validation failure.

A 500 database-corruption response occurs when a resource that is retrieved from the database fails to conform to the schema expected of it by the application. This usually indicates a software bug, but can indicate either:

- Genuine corruption in the database
- That a third party has changed values directly in the database

The msg contains a description of how the database corruption could have occurred. The details contains retrieved, schema, and error keys, which report retrieved resource, the schema it was expected to conform to, and a description of how the resource failed to conform to that schema.

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:

- kind: not-found
- msg: "The resource could not be found"
- details: Contains the URI of the request that caused the 404 response.

Node classifier API v2

These are the endpoints for the node classifier v2 API.

Refer to [Forming node classifier API requests](#) on page 514 for information about calling the node classifier API endpoints.

- [Classification endpoints](#) on page 563

The classification endpoints accept a node name and a set of facts, and then return information about how the specified node is classified. The output can help you test your node group classification rules.

Related information

[API index](#) on page 33

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

Classification endpoints

The classification endpoints accept a node name and a set of facts, and then return information about how the specified node is classified. The output can help you test your node group classification rules.

- [POST /v2/classified/nodes/<name>](#) on page 563

Retrieves classification information for the specified node.

POST /v2/classified/nodes/<name>

Retrieves classification information for the specified node.

Request format

When [Forming node classifier API requests](#) on page 514 to this endpoint, the URI path must contain the name of a specific node, and the body can contain a JSON object using these keys:

Key	Definition
fact	A JSON object containing regular, non-trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.
trusted	A JSON object containing trusted facts associated with the node. The object contains key/value pairs of fact names and fact values. Fact values can be strings, integers, Booleans, arrays, or objects.

Here is an example of a curl command for the /v2/classified/nodes/<name> endpoint:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):4433/classifier-api/v2/
classified/nodes/<NAME>"
data='{"fact" : { "<FACT_NAME>" : "<FACT_VALUE>" }}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

Response format

A successful response returns a JSON object using these keys to describe the node's classification:

Key	Definition
groups	An array of the groups that the node was classified into. Each group is represented by an object containing the group id and the name. Contents of the array are sorted by group name.
environment	The name of the environment that the node uses, which is taken from the node groups the node was classified into.
classes	An array of strings representing the classes that this node received from the groups it was classified into.
parameters	An object containing key/value pairs describing class parameter values for the node's classes if the parameters are different from the default parameters. Each key/value pair consists of a class name, as a string, and a subsequent object containing the names and values of non-default parameters within the named class.

For example:

```
{
  "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 there is an error, [Node classifier API errors](#) on page 562 provide error information in the kind key.

If the node is classified into multiple node groups that supply conflicting classifications to the node, the server returns a 500 Server error response.

For classification-conflict errors, the msg describes generally why the conflict happened, and the details contains an object that uses the environment, variables, or classes key to indicate the type of conflict (whether it was in setting the environment, setting variables, or setting class parameters). Each key contains value-detail objects describing the specific conflicts:

Value-detail object key	Definition
value	The specific value having a conflict. For environment and classes, these are strings. For variables, these can be any JSON value type.

Value-detail object key	Definition
from	The node group that the node was classified into that caused the conflicting value to be added to the node's classification. Refer to <code>defined_by</code> for further details.
defined_by	The node group that actually defined the conflicting value. This is often the <code>from</code> group, but could be an ancestor of that group, due to How node group inheritance works on page 440.

The following example demonstrates a conflicting value being inherited from an ancestor group and a conflicting value supplied directly from the assigned node group. The conflicting value Blue Suede Shoes was included in the classification because the node matched the Elvis Presley group (as indicated by `from`). However, the conflicting value was actually defined by the Carl Perkins group, which is an ancestor of the Elvis Presley group. This caused the child group to inherit the value from the ancestor group. The Since You've Been Gone conflicting value is defined by the same group that the node was assigned to.

```
{
  "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" } },
              ...
            }
          }
        ]
      }
    }
  }
}
```

Node inventory API v1

These are the endpoints for the node inventory v1 API.

- [Forming node inventory API requests](#) on page 566

Requests to the node inventory service API must be well-formed HTTP(S) requests.

- [Command endpoints](#) on page 567

Use the command endpoints to create and delete connections in the inventory service database.

- [Query endpoints](#) on page 571

Use the query endpoints to retrieve lists of inventory service connections.

- [Node inventory API errors](#) on page 574

Node inventory API error responses are formatted as JSON objects.

Related information

[API index](#) on page 33

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

Forming node inventory API requests

Requests to the node inventory service API must be well-formed HTTP(S) requests.

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`.

Node inventory API requests must include a URI path following the pattern:

```
https://<DNS>:8143/inventory/v1/<ENDPOINT>
```

The variable path components derive from:

- **DNS:** Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).
- **ENDPOINT:** Multiple sections specifying the endpoint, such as `command/create-connection` or `query/connections`.

For example, you could use any of these paths to call the [GET /query/connections](#) on page 571 endpoint:

```
https://$(puppet config print server):8143/inventory/v1/query/connections
https://localhost:8143/inventory/v1/query/connections
https://puppet.example.dns:8143/inventory/v1/query/connections
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Token authentication

You must authenticate node classifier API requests. You do this by supplying user authentication tokens in an X-Authentication request header.

For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

This example uses a token generated with `puppet-access login` to call the [POST /command/create-connection](#) on page 567 endpoint:

```
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"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

This example uses the same token and header pattern to call the [POST /query/connections](#) on page 573 endpoint:

```
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 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

Command endpoints

Use the command endpoints to create and delete connections in the inventory service database.

POST /command/create-connection

Create a new connection entry in the node inventory service database.

Request format

Connection database entries contain connection settings, such as certnames, transport methods, and credentials, that are used to connect to nodes (identified by their certnames).

When [Forming node inventory API requests](#) on page 566 to this endpoint, the request body must be a JSON object that uses these required keys:

- `certnames`: An array containing a list of certnames to associate with the supplied connection details.
- `type`: A string that is either `ssh` or `winrm`. This tells `bolt-server` which connection type to use to access the node when running a task.

- **parameters**: An object containing key/value pairs specifying the connection parameters for the specified transport type. Required and optional parameters depend on the transport method and are described further below.

Important: When the Puppet orchestrator targets a certname to run a task, it first considers the value of the `hostname` key present in the connection `parameters`, if supplied. Otherwise, it uses the value of the `certnames` key as the hostname. Make sure to include the `hostname` key only when the hostname differs from the `certname`. If you're configuring multiple connections (`certnames`) at once, do not include a `hostname` key.

- **sensitive_parameters**: An object containing key/value pairs defining the necessary sensitive data for connecting to the provided certnames, such as usernames and passwords. These values are stored in an encrypted format. Required and optional parameters depend on the transport method and are described further below.
- **duplicates**: A string that is either `error` or `replace`. This specifies how to handle cases where supplied `certnames` conflict with existing certnames stored in the node inventory connections database. If you specify `error`, the endpoint returns a 409 response if it finds any duplicate certnames. If you specify `replace`, the endpoint overwrites the existing certname with the new connection details if it finds a duplicate.

Here is an example of a complete request to the `/command/create-connection` endpoint that specifies an SSH connection:

```
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"
```

SSH parameters and sensitive parameters

When the connection `type` is `ssh`, the following keys are available for the `parameters` object. Only the `user` key is required.

user

Required. A string naming the user to log in as when connecting to the host.

port

An integer defining the connection port.

Default: 22

connect-timeout

An integer specifying the length of time, in seconds, that you want PE to wait when establishing connections.

run-as

A string specifying the user name to use to run commands after logging in to the host.

Default: The same value as user.

tmpdir

A string specifying the directory to use to upload and execute temporary files on the target.

Specify this only if the temporary directory is different than /temp.

tty

A Boolean specifying whether to enable text terminal allocation.

hostname

A string specifying the hostname to connect to if it is different from the certname.

When the Puppet orchestrator targets a certname to run a task, it first considers the value of the hostname key present in the connection parameters, if supplied. If omitted, the certname is used as the hostname when the orchestrator connects to the host.

Important: Do not specify a hostname when configuring multiple connections (multiple certnames) in the same request.

Only specify hostname if the node's certname in PE is different than the hostname of the intended target. In this case the certname in the request body must be the desired node certname, and the hostname in the parameters object must be the hostname to connect to.

When the connection type is ssh, the following keys are available for the sensitive-parameters object:

password

Conditionally required. A string specifying the password to use to authenticate the connection.

To form a valid request, you must specify either password or private-key-content.

private-key-content

Conditionally required. The contents of a private key, as a string.

To form a valid request, you must specify either password or private-key-content.

sudo-password

An optional string specifying the password to use when changing users via run-as.

Only include this if run-as is specified in the parameters object.

WinRM parameters and sensitive parameters

When the connection type is winrm, the following keys are available for the parameters object. Only the user key is required.

user

Required. A string naming the user to log in as when connecting to the host.

port

An integer defining the connection port.

Default: 22

connect-timeout

An integer specifying the length of time, in seconds, that you want PE to wait when establishing connections.

tmpdir

A string specifying the directory to use to upload and execute temporary files on the target.

Specify this only if the temporary directory is different than /temp.

extensions

An array listing file extensions that are allowed for tasks.

hostname

A string specifying the hostname to connect to if it is different from the `certname`.

When the Puppet orchestrator targets a `certname` to run a task, it first considers the value of the `hostname` key present in the connection `parameters`, if supplied. If omitted, the `certname` is used as the `hostname` when the orchestrator connects to the host.

Important: Do not specify a `hostname` when configuring multiple connections (multiple `certnames`) in the same request.

Only specify `hostname` if the node's `certname` in PE is different than the `hostname` of the intended target. In this case the `certname` in the request body must be the desired node `certname`, and the `hostname` in the `parameters` object must be the `hostname` to connect to.

When the connection `type` is `winrm`, the `sensitive-parameters` object allows only one key, which is the `password` key. This key required and contains a string specifying the password to use to authenticate the connection.

Response format

If the request is well-formed, valid, and the entries are successfully recorded in the database, the server returns a 201 response with a JSON object containing the `connection_id` for each connection's record. For example:

```
{
  "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec"
}
```

The endpoint also inserts each of the provided `certnames` into PuppetDB with an empty fact set, if they are not already present. After `certnames` are added to PuppetDB, you can view them from the **Nodes** page in the Puppet Enterprise (PE) console. You can also add them to your inventory node lists when you set up jobs to run tasks.

Error responses

Error responses follow the usual format of [Node inventory API errors](#) on page 574.

POST /command/delete-connection

Remove specified `certnames` from all associated connection entries in the inventory service database. In PuppetDB, removed `certnames` are replaced with `preserve: false`.

Request format

When [Forming node inventory API requests](#) on page 566 to this endpoint, the request body must be a JSON object containing the `certnames` key. This key is an array of `certnames` you want to remove. For example:

```
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": [ "mynode5", "mynode6" ] }'

curl --insecure --header "$type_header" --header "$auth_header" --request
  POST "$uri" --data "$data"
```

Response format

If the request is well-formed, valid, and processed successfully, the service returns a 204 response with an empty body.

Error responses

Error responses follow the usual format of [Node inventory API errors](#) on page 574. If you are not authorized to delete connections, the service returns a 403 response.

Query endpoints

Use the query endpoints to retrieve lists of inventory service connections.

GET /query/connections

List all the connections entries in the inventory database or request information about a specific connection.

Request format

When [Forming node inventory API requests](#) on page 566 to this endpoint, the request is a basic GET call with authentication. You can append these optional parameters to the URI path:

- `certname`: A single certname, as a string. Use this to retrieve an individual node's connection details, rather than details for all nodes.
- `sensitive`: A Boolean indicating whether you want the response to include sensitive connection parameters. This parameter has a permission gate, and it doesn't work if you don't have the proper permissions.
- `extract`: Array of keys indicating the information you want the response to include. The `connection_id` key is always returned, and you can use `extract` to limit the remaining keys. For example, `extract=["type"]` limits the response to `connection_id` and `type`.

Tip: To return sensitive parameters, the request must include `sensitive=true`. Otherwise, sensitive parameters are excluded by default.

For example:

```
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'&sensitive=true"

curl --insecure --header "$type_header" --header "$auth_header" "$uri"
```

Response format

A successful response returns a JSON object containing the `items` key. The `items` key is an array of objects, where each object represents a known connection (or a connection for a single node, depending on the request format). The connection objects use these keys:

- `connection_id`: A string that is the unique identifier for the connections entry.
- `certnames`: Array of strings that are the certnames of the matching connections entries.
- `type`: A string describing the connection type, such as `ssh` or `winrm`.
- `parameters`: An object containing arbitrary key/value pairs that describe connection settings.
- `sensitive_parameters`: If specified in the request, and the requesting user has permission to access this information, this key is an object that contains arbitrary key/value pairs describing the connections sensitive settings.

For example, this response describes four connections. This is the typical format to expect when your request includes no additional parameters:

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["node.a", "node.b"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    },
    {
      "connection_id": "4932bfe7-69c4-412f-b15c-ac0a7c2883f1",
      "certnames": ["mynode1", "mynode2"],
      "type": "winrm",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```

This response describes a connection for a specific certname:

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["my.node"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      }
    }
  ]
}
```

This response describes a specific certname and includes the sensitive information:

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["my.node"],
      "type": "ssh",
      "parameters": {
        "tmpdir": "/tmp",
        "port": 1234
      },
      "sensitive_parameters": {
        "username": "<USERNAME>",
        "password": "<PASSWORD>"
      }
    }
  ]
}
```

This response describes a specific aspect of a specific connection (because `certname` and `extract` were supplied in the request):

```
{
  "items": [
    {
      "connection_id": "3c4df64f-7609-4d31-9c2d-acfa52ed66ec",
      "certnames": ["my.node"],
      "type": "ssh"
    }
  ]
}
```

POST /query/connections

Retrieve connection details for a set of certnames.

Request format

When [Forming node inventory API requests](#) on page 566 to this endpoint, the request body must be a JSON object. At minimum, it must be an empty object ({}), or it can use these keys:

- `certnames`: An array containing a list of certnames to retrieve from the inventory service database. If omitted, then all connections are returned.
- `sensitive`: An optional Boolean indicating whether you want the response to include sensitive connection parameters. This parameter has a permission gate, and it doesn't work if you don't have the proper permissions.
- `extract`: An array of keys indicating the information you want the response to include. The `connection_id` key is always returned, and you can use `extract` to limit the remaining keys. For example, ["type"] limits the response to `connection_id` and `type`. If omitted, all keys are returned.

Tip: To return sensitive parameters, the request must include "sensitive": "true". Otherwise, sensitive parameters are excluded by default.

Here are some examples of JSON bodies for the /query/connections endpoint.

An empty request body, which returns information for all known connections but does not include sensitive parameters:

```
{}
```

A request for connection details for a specific certname:

```
{
  "certnames": [ "mynode1" ]
}
```

A request for a specific certname, specific keys, and sensitive parameters:

```
{
  "certnames": [ "averygood.device" ],
  "sensitive": "true",
  "extract": [ "certnames", "sensitive_parameters" ]
}
```

Response format

The successful response is the same as the [GET /query/connections](#) on page 571 endpoint.

Node inventory API errors

Node inventory API error responses are formatted as JSON objects.

Error responses use these keys:

Key	Definition
kind	The kind of error encountered.
msg	The message associated with the error.,
details	A hash containing additional information about the error. This information might be less user-friendly than the msg.

Common errors include:

500 unknown-error

An unknown error occurred.

406 not-acceptable

The request contains an `accepts` header that doesn't allow JSON.

416 unsupported-type

The request contains a `content-type` header that is not JSON.

400 json-parse-error

There is a problem in the request body.

400 schema-validation-error

The body contains data that isn't in the expected or required format.

403 not-permitted

The user submitting the request doesn't have permission to perform the requested action.

This response is only expected for [Command endpoints](#) on page 567.

409 duplicate-certnames

The `duplicates` parameter is not specified, or it is specified as `error`, and one or more of the certnames in the request body already exist in the inventory.

This response is only expected for [Command endpoints](#) on page 567.

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 575

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 581

After configuring patch management, you can start applying patches to nodes. The `patch_server` task enables simply applying patches, while the `group_patching` plan performs health checks before and after patches are applied.

Configuring patch management

To enable patch management, create a node group for nodes you want to patch and add the node group to the **PE Patch Management** parent node group.

Patch management OS compatibility

Patch management is compatible with current Linux operating systems using YUM, APT, and Zypper package management, as well as Microsoft Windows operating systems. We currently test against the following platforms, and these are confirmed to be compatible.

Operating system	Versions
AlmaLinux	8
Amazon Linux	2
CentOS	7
Debian	10, 11
Fedora	36
Note: You must install cron to run patch management on Fedora. To install cron, run <code>dnf install cronie</code>	
Microsoft Windows	10, 11
Microsoft Windows Server	2012, 2012 R2, 2016, 2019, 2022
Note: You must use PowerShell 3.0 or higher to patch Windows nodes.	
Oracle Linux	7, 8
Red Hat Enterprise Linux	7, 8, 9
Rocky Linux	8
Scientific Linux	7
SUSE Linux Enterprise Server	12, 15
Ubuntu	18.04, 20.04, 22.04

Note: If your operating system doesn't support TLSv1.2 or higher, you must [Enable TLSv1 or 1.1](#).

Where patch information comes from

Your package management software is responsible for ensuring PE can find the latest patch information available.

The `pe_patch` module uses OS level tools or APIs to find patches for nodes. You still have to manage the configuration of your package manager, like YUM, APT, Zypper, WSUS, or Windows Update, so your nodes can search for updates. For example, if you need to go through a proxy and you use YUM, you must configure this on your own.

Patching involves two distinct steps. First, a cron job scans for new patches and uploads related details to PuppetDB as part of the `pe_patch` fact. You can specify when to run the cron job with parameters in the `pe_patch` class. Then, patches are applied to specified nodes using the `pe_patch::patch_server` task or the `pe_patch::group_patching` plan.

Note: If you need to restrict which packages/patches your OS finds and which patches are applied:

- For *nix agents patching: Pin a package using `yum versionlock`, `apt-mark`, or `zypper addlock`. The `pinned_packages` field in the `pe_patch` fact refers to versions locked using these methods. This is different from `apt-pinning` packages, which is used to prioritize packages rather than locking them at a specific version.
- For Windows agents patching: If you use WSUS or Windows Update to deliver updates, use WSUS to approve desired updates independently.

Security updates

To find security updates, the `pe_patch` module uses security metadata when it is available. For example, Red Hat provides security metadata as additional metadata in YUM, Debian performs checks on the repo the updates are coming from, and Windows provides this information by default.

In the console, on the **Patches** page, security metadata feeds into the **Apply patches** table where you can filter for **Security updates only**.

Configure Windows Update

If you are using Windows Update, we recommend you use the [puppetlabs/wsus_client](#) module and configure these parameters in the `wsus_client` class.

- Set the `server_url` parameter to the URL of your WSUS server.
- Set the `auto_update_options` parameter to `AutoNotify` to automatically download updates and notify users.

Create a node group for nodes under patch management

Create a node group for nodes you want to patch in Puppet Enterprise (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.



CAUTION: 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.

- 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 can cause classification conflicts.

- Select **Run > Puppet**.

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 442

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.

- On the **Node groups** page, select the patching node group you want to add parameters to.
- If it doesn't already exist, add the **pe_patch** class to the node group.
 - On the **Classes** tab, enter **pe_patch** and select **Add class**.
 - Commit changes.
- 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.

- Specify any additional patching parameters in the **pe_patch** class.
- 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 577 for more information.

- On the **Node groups** page, select the patching node group you want to assign a blackout window to.
- On the **Classes** tab, under **Parameter**, add the **blackout_windows** parameter to the **pe_patch** class.

- 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"
  }
}
```

- Commit changes.

When a user tries to patch nodes during the blackout window, the **Patch blocked** field on the **Apply patches** table changes from **No** to **Yes** for affected patches. If the user proceeds with patching, the patching task fails.

Patch management parameters

Configure and tune patch management by adjusting parameters in the `pe_patch` class.

patch_data_owner

User name for the owner of the patch data. String.

Default: `root`

patch_data_group

Group name for the owner of the patch data. String.

Default: `root`

patch_cron_user

User account for running the cron job that scans for new patches in the background. String.

Default: `$patch_data_owner`

manage_yum_utils

Determines if the `yum_utils` package should be managed by this module on RedHat family nodes. If `true`, use the `yum_utils` parameter to determine how it should be managed. Boolean.

Default: `false`

yum_utils

If managed, determines what the package is set to. Enum[`installed`, `absent`, `purged`, `held`, `latest`]

Default: `installed`

block_patching_on_warnings

Determines if the patching task should run if there were warnings present on the `pe_patch` fact. If `true`, the run will abort and take no action. If `false`, the run will continue and attempt to patch. Boolean.

Default: `false`

fact_upload

Determines if `puppet fact upload` runs after any changes are made to the fact cache files. Boolean.

Default: `true`

apt_autoremove

Determines if `apt-get autoremove` runs during reboot. Boolean.

Default: `false`

manage_delta_rpm

Determines if the `delta_rpm` package should be managed by this module on RedHat family nodes. If `true`, use the `delta_rpm` parameter to determine how it should be managed. Boolean.

Default: false

delta_rpm

If managed, determines what the `delta_rpm` package is set to. Enum[`installed`, `absent`, `purged`, `held`, `latest`]

Default: `installed`

manage_yum_plugin_security

Determines if the `yum_plugin_security` package should be managed by this module on RedHat family nodes. If true, use the `yum_plugin_security` parameter to determine how it should be managed. Boolean.

Default: false

yum_plugin_security

If managed, determines what the `yum_plugin_security` package is set to. Enum[`installed`, `absent`, `purged`, `held`, `latest`]

Default: `installed`

reboot_override

Determines if a node reboots after patching. This overrides the setting in the task. Variant, Boolean, Enum[`always`, `never`, `patched`, `smart`, `default`]

- `always` - The node always reboots during the task run, even if no patches are required.
- `never` (or `false`) - The node never reboots during the task run, even if patches are applied.
- `patched` (or `true`) - The node reboots if patches are applied.
- `smart` - Use the OS supplied tools, like `needs_restarting` on RHEL or a pending reboot check on Windows, to determine if a reboot is required, if it is reboots, or if it does not reboot.
- `default` - Uses whatever option is set in the `reboot` parameter for the `pe_patch::patch_server` task.

Default: `default`

patch_group

Identifies nodes in or across patching node groups to run patching plans against.

Default: `undef`

pre_patching_scriptpath

The full path to an executable script or binary on the target node to be run before patching.

Default: `undef`

post_patching_scriptpath

The full path to an executable script or binary on the target node to be run after patching.

Default: `undef`

patch_cron_hour

The hour or hours to run the cron job that scans for new patches.

Default: `absent`, or *

patch_cron_month

The month or months to run the cron job that scans for new patches.

Default: `absent`, or *

patch_cron_monthday

The monthday or monthdays to run the cron job that scans for new patches.

Default: `absent`, or *

patch_cron_weekday

The weekday or weekdays to run the cron job that scans for new patches.

Default: absent, or *

patch_cron_min

The min or mins to run the cron job that scans for new patches.

Default: fqdn_rand(59) - a random number between 0 and 59.

ensure

Use present to install scripts, cronjobs, files, etc. Use absent to clean up system that previously hosted.

Default: present

blackout_windows

Determines a window of time when nodes cannot be patched. Hash.

:title - Name of the blackout window. String.

:start - Start of the blackout window (ISO8601 format). String.

:end - End of the blackout window (ISO8601 format). String.

Default: undef

windows_update_criteria

Determines which types of updates Windows Update searches for. To search both software and driver updates, remove the Type argument. String.

Default: IsInstalled=0 and IsHidden=0 and Type='Software'

Note: See the [Microsoft documentation](#) for more information about formatting strings for Windows Update.

Disable patch management

Use the console to disable patch management by editing the ensure parameter in the **PE Patch Management** node group. You can also remove patch management by deleting patching node groups.

1. In the console, click **Node groups** and select the **PE Patch Management** node group.
2. On the **Classes** tab, under the `pe_patch` class, select the `ensure` parameter, and change the value to `absent`.
3. Click **Add to node group** and commit the change.
4. Run Puppet.
The client components of the `pe_patch` class, like cron and scripts, are removed from PE.
5. Optional: To remove patch management from your infrastructure, click **Remove node group** on the **Node details** page for the **PE Patch Management** node group.

Note: If you have any child node groups under patch management, you must remove those node groups prior to removing the **PE Patch Management** parent node group.

The **Patch Management** section in the console sidebar remains active after disabling patch management, but the **Patches** page no longer reports patch information.

Patching nodes

After configuring patch management, you can start applying patches to nodes. The `patch_server` task enables simply applying patches, while the `group_patching` plan performs health checks before and after patches are applied.

Patch nodes

Use the `patch_server` task to apply patches to nodes. You can limit patches to security or non-security updates, Windows or *nix nodes, or a specific patch group.

Before you begin

Ensure you have permission to run the `pe_patch::patch_server` task.

- 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.

- Select **Run > Task**.

The **Run a task** page appears with patching information pre-filled for the `pe_patch::patch_server` task.

- Optional: In the **Job details** field, provide a description of the task run. This text appears on the **Tasks** page.
- Optional: Under **Task parameters**, add optional parameters to the task. See [Patching task parameters](#) on page 581 for a full list of available parameters.

Note: You must click **Add parameter** for each optional parameter-value pair you add to the task.

- Optional: If you want to schedule the task to run later, under **Schedule**, select **Later** and choose a time.
- 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`.

- 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 577.

- Optional: In the **Schedule** section, specify if you want the plan to run at a later date and time.

- Click **Run job**.

To check the status of the plan, look for it on the **Plans** page.

Note: When using patch management to update core packages that affect the networking stack, the task run might look like it failed due to the PXP agent on the node losing connection with the primary server. However, the task still completes successfully. You can confirm by checking the **pe_patch** fact to verify the relevant packages were updated.

Patching plan parameters

The `pe_patch::group_patching` plan verifies that Puppet is configured and running correctly on target nodes, patches the nodes, waits for any reboots, and then runs Puppet on the nodes to verify that they're still operational.

By default, the plan includes a health check which considers "healthy" any nodes on which:

- The Puppet service is enabled and running
- Noop mode and cached catalogs are not enabled
- The run interval is 30 minutes

You can modify plan behavior with several types of optional parameters:

- Patching options let you control how patching itself is applied, including adding an optional string to arguments passed to your package provider.
- Health check options control when a pre-patching health check and a post-patching Puppet run occurs.

Tip: The `health_check_*` parameters apply patches only to nodes that match values you specify. For example, if you change `health_check_service_running` to `false`, the pre-patching health check marks nodes on which the Puppet service *is* running as "unhealthy" and skips patching them.

- Reboot options control when a post-patching reboot occurs, and let you specify a script to execute after patching.

Patching options

patch_group

Specifies the **patch_group**, as defined in the **pe_patch** class parameter, that you want to base running the plan on.

Accepted values: String

patch_task_timeout

Indicates how much time elapses before the task run times out.

Accepted values: Any positive integer, in seconds.

Default: 3600

security_only

Indicates whether to apply only security patches.

Accepted values: `true`, `false`

Default: `false`

yum_params

Indicates additional parameters to include in YUM commands, such as including or excluding repositories.

Accepted values: String

Default: undef

dpkg_params

Indicates additional parameters to include in apt-get commands.

Accepted values: String

Default: undef

zypper_params

Indicates additional parameters to include in Zypper commands.

Accepted values: String

Default: undef

clean_cache

Indicates if YUM or dpkg caches are cleaned at the start of the task.

Accepted values: true, false

Default: false

sequential_patching

Indicates if nodes in the specified patch group are patched, rebooted, and the post-reboot script run one at a time rather than all at once.

Accepted values: true, false

Default: false

Health check options

run_health_check

Indicates whether to do a pre-patching health check and a post-patching Puppet run.

Accepted values: true, false

Default: true

health_check_noop

Verifies the noop setting during pre-patching health checks.

Accepted values: true, false

Default: false

health_check_runinterval

Verifies the runinterval setting during pre-patching health checks.

Accepted values: Any positive integer, in seconds.

Default: 1800 (equivalent to the default Puppet run interval of 30 minutes)

health_check_service_running

Verifies whether the Puppet service is running during pre-patching health checks.

Accepted values: true, false

Default: true

health_check_service_enabled

Verifies whether the Puppet service is enabled during pre-patching health checks.

Accepted values: true, false

Default: true

health_check_use_cached_catalog

Verifies the use_cached_catalog setting during pre-patching health checks.

Accepted values: true, false

Default: false

Reboot options

`reboot`

Indicates if and when nodes reboot during the plan run.

Note: If the node group you're patching has a `reboot_override` value specified, that value overrides any `reboot` parameter you specify in plan runs.

Accepted values:

- always — The node always reboots during the plan run, even if no patches are required.
- never — The node never reboots during the plan run, even if patches are applied.
- patched — The node reboots if patches are applied.
- smart — Use the OS supplied tools, like `needs_restarting` on RHEL or a pending reboot check on Windows, to determine if a reboot is required.

Default: patched

`reboot_wait_time`

Indicates how long to wait for nodes to reboot before running a post-patching health check.

Accepted values: Any positive integer, in seconds.

Default: 600

`post_reboot_scriptpath`

The full path to an executable script or binary on the target node to be run after reboot and before the final Puppet run.

Accepted values: File path

Default: undef

Orchestrating Puppet runs, tasks, and plans

Puppet orchestrator is an effective tool for making on-demand changes to your infrastructure.

With orchestrator you can initiate Puppet, task, or plan runs whenever you need them, eliminating manual work across your infrastructure.

- [How Puppet orchestrator works](#) on page 586

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

- [Setting up the orchestrator workflow](#) on page 590

The orchestrator—used alongside other Puppet Enterprise (PE) tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

- [Configuring Puppet orchestrator](#) on page 597

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.

- [Run Puppet on demand](#) on page 604

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

- [Tasks in PE](#) on page 614

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 645

Plans allow you to tie together tasks, scripts, commands, and other plans to create complex workflows with refined access control. You can install modules that contain plans or write your own, then run them from the console or the command line.

- [Orchestrator API v1](#) on page 678

You can use the orchestrator API to run jobs and plans on demand; schedule tasks and plans; get information about jobs, plans, and events; track node usage; and more.

- [Migrating Bolt tasks and plans to PE](#) on page 759

If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a Puppet Enterprise (PE) environment. This lets you manage and run tasks and plans using PE and the console. Bolt projects have the same structure as Puppet modules, and they can be loaded from the `modules` directory of a PE environment.

How Puppet orchestrator works

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

When you run Puppet on-demand with the orchestrator, you control the rollout of configuration changes when and how you want them. You control when Puppet runs and where node catalogs are applied (from the environment level to an individual node). You no longer need to wait on arbitrary run times to update your nodes.

Puppet tasks allow you to execute actions on target machines. A "task" is a single action that you execute on the target via an executable file. For example, do you want to upgrade a package or restart a particular service? Set up a Puppet task run to enforce to make those changes at will.

Puppet plans are bundles of tasks that can be combined with other logic. They allow you to do complex operations, like run multiple tasks with one command or automatically run certain tasks based on the output of another task.

Tasks and plans are packaged and distributed as Puppet modules.

Puppet orchestrator technical overview

The orchestrator uses `pe-orchestration-services`, a JVM-based service in Puppet Enterprise (PE), to execute on-demand Puppet runs on agent nodes in your infrastructure. The orchestrator uses Puppet Execution Protocol (PXP) agents to orchestrate changes across your infrastructure.

The orchestrator (part of `pe-orchestration-services`) controls the functionality for the `puppet job`, `puppet task`, and `puppet plan` commands, and it 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 orchestrator's functionality derives from the Puppet Execution Protocol (PXP), the Puppet Communications Protocol (PCP), and the Agentless Catalog Executor (ACE) Server.

Puppet Execution Protocol (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.

Used by `pe-orchestration-services` to run Puppet on agents.

PXP agent

A system service in the agent package that runs PXP.

Puppet Communications Protocol (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 (which includes the primary server, PuppetDB, and the PE console) connect directly to the orchestrator, but all other PXP agents connect to compilers via load balancers.

Agentless Catalog Executor (ACE) service

A Ruby service that enables you to execute tasks, plans, and Puppet runs on remotely on agentless targets. Refer to [PE ACE server configuration](#) on page 603 for more information.

Bolt vs ACE: Orchestrator uses both ACE and Bolt to run tasks and plans. While both can act on agentless targets, the primary difference is that Bolt server works with agentless nodes over WinRM or SSH, whereas ACE works with agentless devices, like network switches and firewalls, over other transports. Go to [PE Bolt server configuration](#) on page 602 to learn about how Bolt works in PE and configuring the Bolt server.

What happens during an on-demand run from the orchestrator ?

Several PE services interact when you [Run Puppet on demand](#) on page 604 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. It also queries PuppetDB for the nodes.
4. The orchestrator creates the job ID and starts polling nodes in the job to check their statuses.
5. The orchestrator queries PuppetDB for the agent version on the nodes in the job.
6. The orchestrator tells the PCP broker to start runs on the nodes in the job, and Puppet runs start on those agents.
7. Agents send run results to the PCP broker.
8. The orchestrator receives run results and requests node run reports (also called agent run reports) from PuppetDB.

What happens during a task run from the orchestrator?

Several services interact during task runs. 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 runs, the orchestrator sends the PXP agent a URL indicating where to fetch the task from (on the primary server) and the task file's checksum. The PXP agent downloads the task file from the supplied URL and caches it for future use. The file is validated against the checksum before every execution. This process is comprised of the following steps:

1. The PE client sends a task command.
2. The orchestrator checks if the user is authorized.
3. The orchestrator fetches the node target from PuppetDB (if the target is a query) and returns the list of targeted 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.

Notes about configuring the orchestrator and related components

Various files contain configuration and tuning settings for orchestrator components.

pe-orchestration-services

This is the underlying service for the orchestrator.

The main configuration file is located at: `/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 located at: `/etc/puppetlabs/puppetserver/conf.d`

The PCP broker requires JVM memory and file descriptors. 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 example, for a deployment with 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 in the agent profile class (`puppet_enterprise::profile::agent`).

The PXP agent is configured to use Puppet's SSL certificates and point to one PCP broker endpoint. If you've configured disaster recovery, the agent points to additional PCP broker endpoints in the case of failover.

Important: If you reuse an existing agent with a new orchestrator instance, you must delete the `pxp-agent` spool directory, which is located at:

- `/opt/puppetlabs/pxp-agent/spool` on *nix systems
- `C:\ProgramData\PuppetLabs\pxp-agent\var\spool` on Windows systems

Related information

- [Orchestrator and pe-orchestration-services parameters](#) on page 236
- [Manage ARP table overflow](#) on page 239

- [Java heap](#) on page 207
- [JRuby max active instances](#) on page 206
- [Java parameters](#) on page 140

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 located at:

```
/var/log/puppetlabs/orchestration-services/orchestration-services.log
```

PCP brokers

The main log file for PCP brokers on the primary server is located at:

```
/var/log/puppetlabs/orchestration-services/pcp-broker.log
```

The main log file for PCP brokers on compilers is located at:

```
/var/log/puppetlabs/puppetserver/pcp-broker.log
```

Tip: You can configure logback in your Puppet Server configuration.

You can also enable an access log for messages.

PXP agent

The main log file location varies by OS, and it can be configured if necessary. On *nix systems, the default location is:

```
/var/log/puppetlabs/pxp-agent/pxp-agent.log
```

On Windows systems, the default location is:

```
C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log
```

Metadata about Puppet runs triggered through the PXP agent are kept in the `spool-dir` for 14 days (by default). The `spool-dir` location varies by OS.

- For *nix: `/opt/puppetlabs/pxp-agent/spool`
- For Windows: `C:/ProgramData/PuppetLabs/pxp-agent/var/spool`

ACE server

The main log file is located at:

```
/var/log/puppetlabs/ace-server/ace-server.log
```

Related information

- [Configure PXP agent parameters](#) on page 238

Setting up the orchestrator workflow

The orchestrator—used alongside other Puppet Enterprise (PE) tools, such as Code Manager—allows you to control when and how infrastructure changes are made before they reach your production environment.

This recommended workflow gives you precise control over rolling out changes, such as deploying new Puppet code or updating data and classifying nodes. In this workflow, you configure your agents to use cached catalogs during scheduled runs, and you use orchestrator jobs to send new catalogs only when you're ready. Scheduled runs continue to enforce the desired state from the last orchestration job until you send another new catalog.

Before you begin:

This workflow assumes you're familiar with Code Manager. It involves making changes to your control repo, such as adding or updating modules, editing manifests, or changing your Hiera data.

This workflow requires running deploy actions from the Code Manager command line tool and the orchestrator, so make sure you have access to a host with PE client tools installed.

Related information

[Add code and set up Code Manager](#) on page 74

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

Enable cached catalogs for use with the orchestrator

Enabling cached catalogs on your agents ensures Puppet does not enforce any catalog changes on your agents until you run an orchestrator job to enforce changes.

When you use the orchestrator to enforce changes in a Puppet environment (for example, in your production environment), you want agents in that environment to maintain their cached catalogs until you run an orchestrator job that deploys configuration changes for those agents. In these environments, agents reinforce configuration from their cached catalogs during the normal run interval (30 minutes by default), and they apply new configuration only when you run Puppet with an orchestration job.

Important: Although enabling cached catalogs is optional (you can run Puppet on nodes with orchestrator workflows that don't require cached catalogs), our recommended workflow requires enabling cached catalogs so agents enforce cached catalogs by default and only compile new catalogs when instructed by orchestrator jobs.

1. Run Puppet on the new agents.



CAUTION: Run Puppet on new agents before assigning any application components to them or performing the next step.

2. In each agent's `puppet.conf` file, add `use_cached_catalog=true` to the `[agent]` section. There are two ways to do this:
 - From the command line on each agent machine, run:

```
puppet config set use_cached_catalog true --section agent
```

- Add an `ini_setting` resource in the `node default {}` section of the environment's `site.pp` file.

Important: This adds the setting to all agents in the environment.

```
if $facts['kernel'] == 'windows' {
  $config = 'C:/ProgramData/PuppetLabs/puppet/etc/puppet.conf'
} else {
  $config = $settings::config
}

ini_setting { 'use_cached_catalog':
  ensure  => present,
  path    => $config,
  section => 'agent',
  setting => 'use_cached_catalog',
  value   => 'true',
}
```

3. Run Puppet on the agents again to enforce this configuration.

Set up node groups for testing new features

1. If they don't already exist, create environment node groups for branch testing. For example, you could create Development environment and Test environment node groups.
2. Within each of these environment node groups, create a child node group to enable on-demand testing of changes deployed in Git feature branch Puppet environments.

You now have at three levels of environment node groups:

- The top-level parent environment node group
 - Node groups representing your actual environments
 - Node groups for feature testing
3. On the **Rules** tab for each feature testing child node group, add a rule with these values:

- **Fact:** `agent_specified_environment`
- **Operator:** `~`
- **Value:** `^.+`

This rule matches any nodes from the parent group that have the `agent_specified_environment` fact set. By matching nodes to this group, you give the nodes permission to override the server-specified environment and use their agent-specified environment instead.

Related information

[Create environment node groups](#) on page 441

Create custom environment node groups so you can target Puppet code deployments.

Create a feature branch

After setting up a node group for feature testing, create a feature branch in your control repository. A feature branch allows you to develop and test code before merging it with the main branch.

1. Create a feature branch in your control repository, and name the branch clearly as a feature branch (for example, `my_feature_branch` or `feature_<TICKET_NUMBER>`).



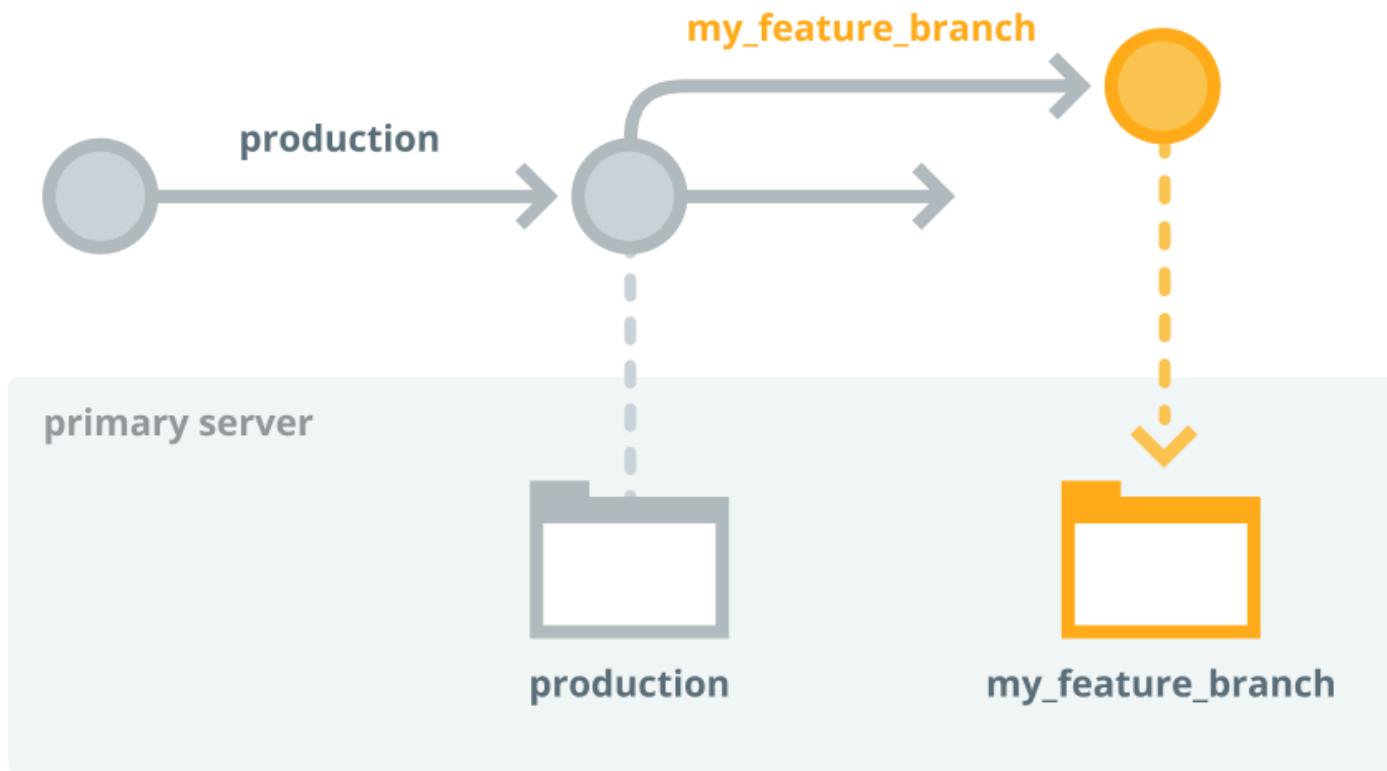
2. Make changes to the code on the feature branch, and commit and push the changes to the feature branch.

Deploy code to the primary server and test it

After making changes to the code on your feature branch, use Code Manager to push those changes to the primary server.

1. To deploy the code from the feature branch to the primary server, run this Code Manager command:

```
puppet code deploy --wait <FEATURE_BRANCH>
```

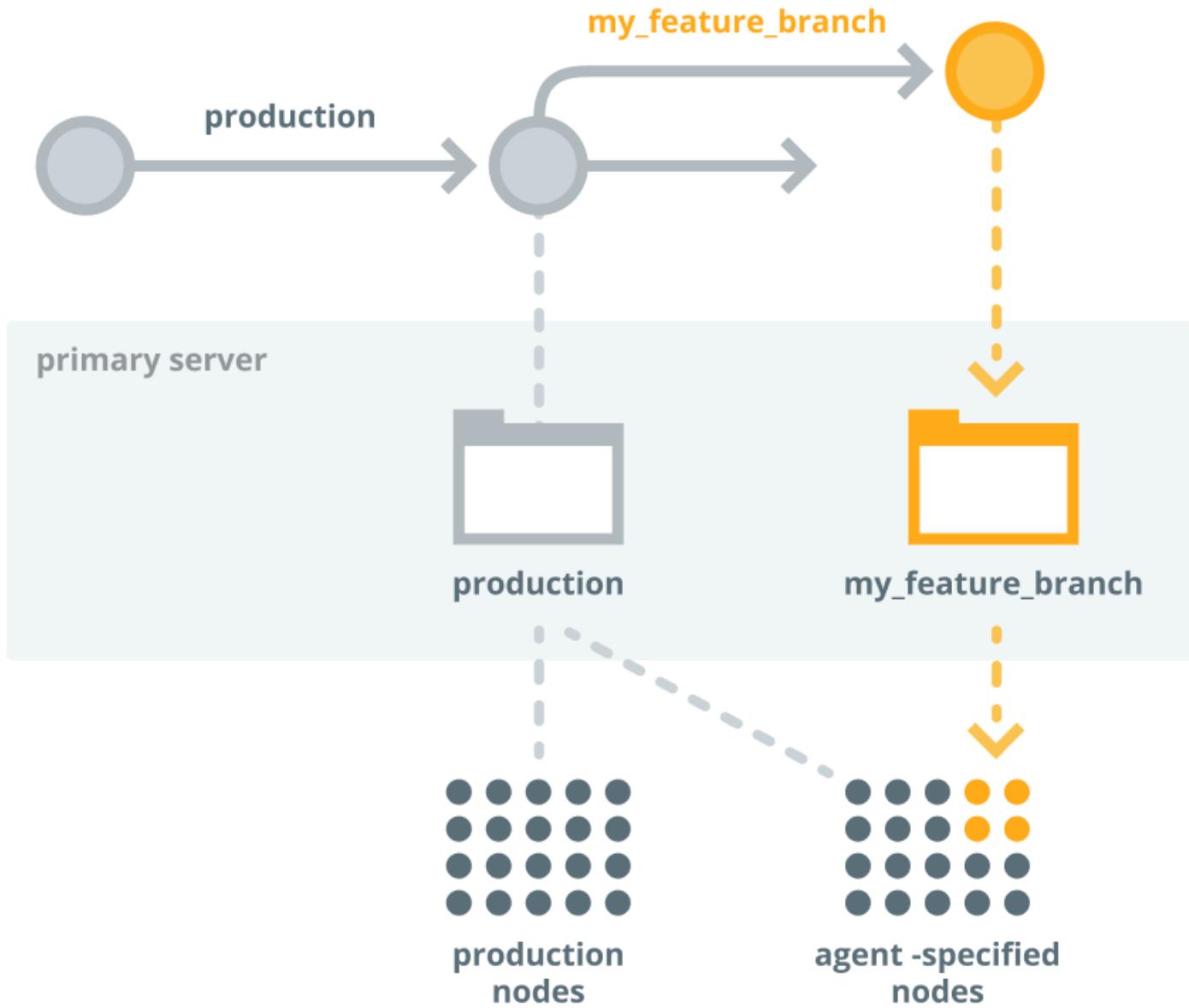


After this code deployment, wait while Puppet Server loads the new code. The primary server now has code from the main/production branch and the feature branch.

- To test your changes, use this orchestrator command to run Puppet on a few agent-specified development nodes in the feature branch environment:

```
puppet job run --nodes <DEV-NODE1>,<DEV_NODE2> --environment
<FEATURE_BRANCH>
```

Tip: You can also use the console to create a job targeting a list of nodes in the feature branch environment.



At this point, the production environment maintains the original code deployed to all production nodes, and the feature branch environment has deployed your new code to the nodes you specified.

- Review the node run reports in the console by opening the links in the orchestrator command output or using the Job ID linked on the **Job list** page. Verify the code changes had the intended effect.

Related information

[Run Puppet on one or more specific nodes](#) on page 605

An orchestrator job can target one or more specific nodes. This is useful if you want to run Puppet on a single node, a few specific nodes, nodes that are not in the same node group, or nodes that can't easily be identified by a single PQL query.

Merge and promote your code

If everything works as expected on the development nodes, you're ready to promote your changes into production.

1. Merge your feature branch into the production branch in your control repo.



2. To deploy your updated production branch to the primary server, run this Code Manager command:

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

Preview the job

Before running Puppet across the production environment, use the `puppet job plan` command to preview the job.

1. To ensure the job captures all nodes in the production environment, as well as the agent-specified development nodes that just ran with the feature branch environment, use this query as the job target:

```
puppet job plan --query 'inventory {environment in ["production",
"<FEATURE_BRANCH>"]}'
```

2. Review the outcome to ensure you want to make these changes.

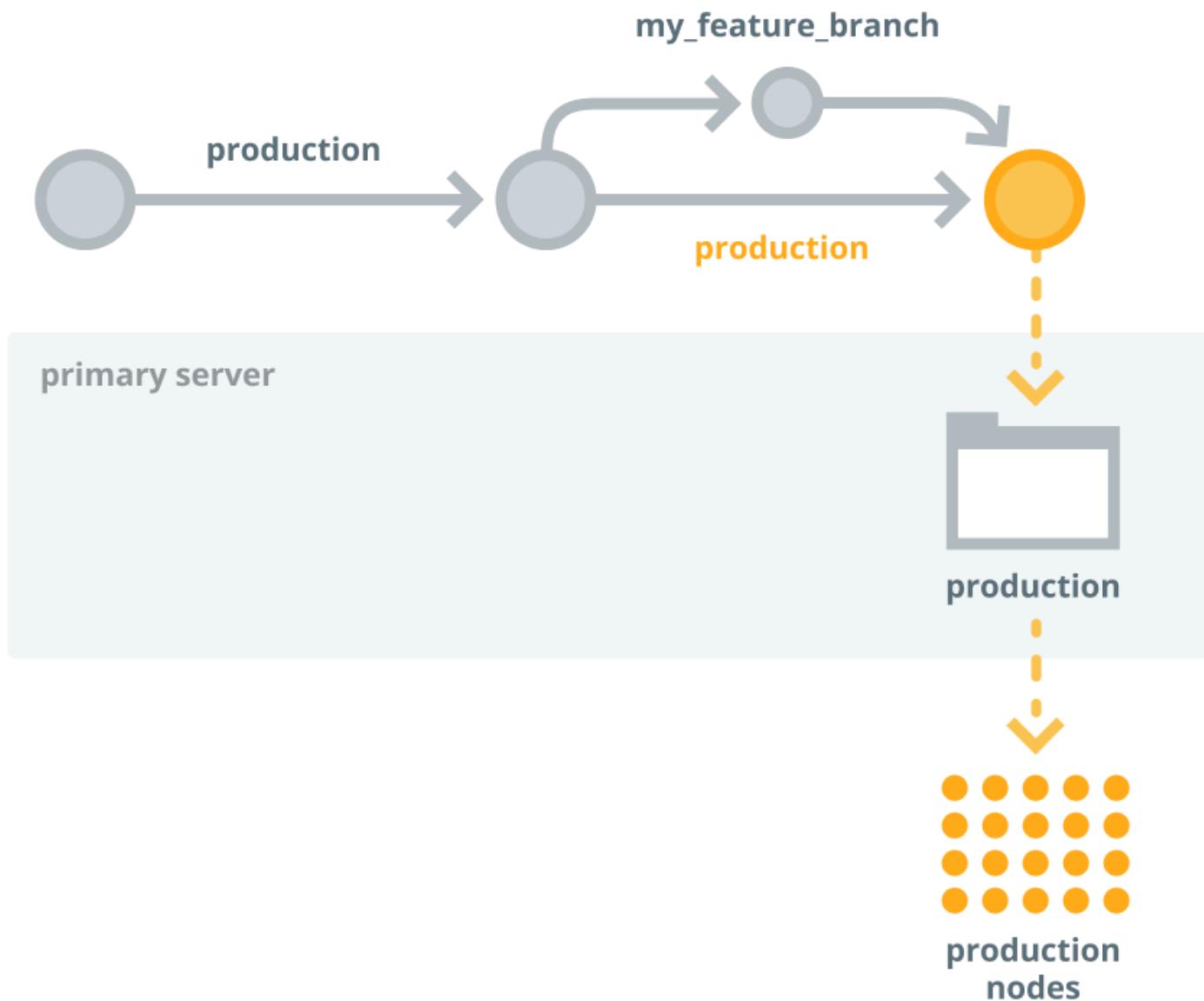
Run the job on the production environment

If you're satisfied with the job preview, you're ready to enforce changes on the production environment.

- To deploy the new code to the primary server and all nodes in the production environment, run the orchestrator job:

```
puppet job run --query 'inventory {environment in ["production", "<FEATURE_BRANCH>"]}'
```

Tip: You can also use the console to create a job targeted at this PQL query.



- Check the node run reports in the console to confirm that the changes were applied as intended.

Repeat this process as you develop and promote your code.

Related information

[Run Puppet on a PQL query](#) on page 605

An orchestrator job can target a set of nodes based on a PQL query. This is useful when you want to target a variable set of nodes that meet specific conditions, such as a particular operating system. When you supply a PQL query, the orchestrator runs the job on a list of nodes generated by the PQL query.

Configuring Puppet orchestrator

Once you've installed PE or the client tools package, there are a few tasks you need to do to prepare your PE infrastructure for orchestration services.

- Set PE RBAC permissions and token authentication for Puppet orchestrator
- Enable cached catalogs for use with the orchestrator (optional)
- Review the orchestrator configuration files and adjust them as needed

All of these instructions assume that PE client tools are installed.

Related information

[Installing client tools](#) on page 172

PE client tools are a set of command line tools that let you access Puppet Enterprise services from a workstation that might or might not be managed by Puppet.

Orchestration services settings

Global logging and SSL settings

`/etc/puppetlabs/orchestration-services/conf.d/global.conf` contains settings shared across the Puppet Enterprise (PE) orchestration services.

The file `global.certs` typically requires no changes and contains the following settings:

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)

Allow list of trapperkeeper services to start

`/etc/puppetlabs/orchestration-services/bootstrap.cfg` is the list of trapperkeeper services from the orchestrator and pcp-broker projects that are loaded when the `pe-orchestration-services` system service starts.

- To disable a service in this list, remove it or comment it with a `#` character and restart `pe-orchestration-services`
- To enable an NREPL service for debugging, add `puppetlabs.trapperkeeper.services.nrepl.nrepl-service/nrepl-service` to this list and restart `pe-orchestration-services`.

The pcp-broker and orchestrator HTTP services

`/etc/puppetlabs/orchestration-services/conf.d/webserver.conf` describes how and where to run pcp-broker and orchestrator web services, which accept HTTP API requests from the rest of the PE installation and from external nodes and users.

The file `webserver.orchestrator` configures the orchestrator web service. Defaults are as follows:

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
client-auth	Determines the mode that the server uses to validate the client's certificate for incoming SSL connections.	want or need
ssl-ca-cert	Sets the path to the CA certificate PEM file used for client authentication.	/etc/puppetlabs/puppet/ssl/certs/ca.pem
ssl-cert	Sets the path to the server certificate PEM file used by the web service for HTTPS.	/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.cert.pem
ssl-crl-path	Describes a path to a Certificate Revocation List file. Optional.	/etc/puppetlabs/puppet/ssl/crl.pem
ssl-host	Sets the host name to listen on for encrypted HTTPS traffic.	0.0.0.0.
ssl-key	Sets the path to the private key PEM file that corresponds with the ssl-cert.	/etc/puppetlabs/orchestration-services/ssl/<orchestrator-host-fqdn>.private_key.pem
ssl-port	Sets the port to use for encrypted HTTPS traffic.	8142

/etc/puppetlabs/orchestration-services/conf.d/web-routes.conf describes how to route HTTP requests made to the API web servers, designating routes for interactions with other services. These should not be modified. See the configuration options at the [trapperkeeper-webserver-jetty project's docs](#)

Analytics trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/analytics.conf contains the internal setting for the [analytics](#) trapperkeeper service.

Setting	Definition	Default
analytics.url	Specifies the API root.	<puppetserver-host-url>:8140/analytics/v1

Authorization trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/auth.conf contains internal settings for the authorization trapperkeeper service. See configuration options in the [trapperkeeper-authorization project's docs](#).

JMX metrics trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/metrics.conf contains internal settings for the JMX metrics service built into orchestration-services. See the service configuration options in the [trapperkeeper-metrics project's docs](#).

Orchestrator trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/orchestrator.conf contains internal settings for the orchestrator project's trapperkeeper service.

PCP broker trapperkeeper service configuration

/etc/puppetlabs/orchestration-services/conf.d/pcp-broker.conf contains internal settings for the pcp-broker project's trapperkeeper service. See the service configuration options in the [pcp-broker project's docs](#).

Related information

[Orchestrator and pe-orchestration-services parameters](#) on page 236

These are some optional parameters you can use to configure the behavior of the orchestrator and the pe-orchestration-services service.

Orchestrator configuration files

The configuration file for the orchestrator allows you to run commands from the CLI without having to pass additional flags. Whether you are running the orchestrator from the primary server or from a separate 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
manage_orchestrator	true or false (default is true)
orchestrator_url	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.

Type	Permission	Definition
Nodes	View node data from PuppetDB.	The ability to view node data imported from PuppetDB. Object must always be " * ".

Note: If you don't have permission to view a node group, or the node group doesn't have any matching nodes, that node group isn't listed as an option. In addition, node groups don't appear if they have no rules specified.

Assign task permissions to a user role

1. In the console, on the **Access control** page, click the **User roles** tab.
2. From the list of user roles, click the one you want to have task permissions.
3. On the **Permissions** tab, in the **Type** box, select **Tasks**.
4. For **Permission**, select **Run tasks**, and then select a task from the **Object** list. For example, **facter_task**.
5. Click **Add permission**, and then commit the change.

Using token authentication

Before running an orchestrator job, you must generate an RBAC access token to authenticate to the orchestration service. If you attempt to run a job without a token, PE prompts you to supply credentials.

For information about generating a token with the CLI, see the documentation on token-based authentication.

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

[Create a user role](#) on page 279

Puppet Enterprise (PE) includes five default roles. You can also create your own roles.

[Assign permissions to a user role](#) on page 279

You can mix and match permissions to create custom user roles that provide users with precise access to Puppet Enterprise (PE) actions.

PE Bolt server configuration

The PE Bolt server provides an API for running tasks over SSH and WinRM using Bolt, which is the technology underlying PE tasks. You do not need to have Bolt installed to configure the Bolt server or run tasks in PE. The API server for tasks is available as `pe-bolt-server`.

Bolt vs ACE: Orchestrator uses both ACE and Bolt to run tasks and plans. While both can act on agentless targets, the primary difference is that Bolt server works with agentless nodes over WinRM or SSH, whereas ACE works with agentless devices, like network switches and firewalls, over other transports.

The PE Bolt 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 the parameters described in the following table:

Setting	Type	Description	Default
bolt_server_loglevelString		Bolt log level. Acceptable values are debug, info, notice, warn, or error.	notice
concurrency	Integer	Maximum number of server threads.	100
master_host	String	URI of the primary server where Bolt can download tasks.	\$puppet_enterprise::puppet_main
master_port	Integer	Port the Bolt server can access the primary server on.	\$puppet_enterprise::puppet_main
ssl_cipher_suites	Array of strings	TLS cipher suites in order of preference.	\$puppet_enterprise::params::ssl_cipher_suites
ssl_listen_port	Integer	Port the Bolt server runs on.	62658 (\$puppet_enterprise::bolt_server_port)
allowlist	Array of strings	List of hosts that can connect to pe-bolt-server.	[\$certname]

Related information

[Configure cipher suites](#) on page 223

Regulatory compliance or other security requirements might require you to change the cipher suites your SSL-enabled PE services use to communicate with other PE components.

PE ACE server configuration

The PE ACE server is a service that allows for tasks and catalogs to run against remote targets that can't run a Puppet agent, such as network switches and firewalls.

Bolt vs ACE: Orchestrator uses both ACE and Bolt to run tasks and plans. While both can act on agentless targets, the primary difference is that Bolt server works with agentless nodes over WinRM or SSH, whereas ACE works with agentless devices, like network switches and firewalls, over other transports.

The ACE server is a [Puma](#) application that runs as a standalone service.

The server is configured in `/etc/puppetlabs/ace-server/conf.d/ace-server.conf` and managed in the `puppet_enterprise::profile::ace_server` class, which includes the parameters described in the following table:

Setting	Type	Description	Default
service_loglevel	String	Bolt log level. Acceptable values are debug, info, notice, warn, or error.	notice
concurrency	Integer	Maximum number of server threads.	\$puppet_enterprise::ace_server

Setting	Type	Description	Default
master_host	String	URI that ACE can access the primary server on.	pe_repo::compile_master_pool Default: \$puppet_enterprise::puppet_ma
master_port	Integer	Port that ACE can access the primary server on.	\$puppet_enterprise::puppet_ma
hostcrl	String	The host CRL path	\$puppet_enterprise::params::h
ssl_cipher_suites	Array of strings	TLS cipher suites in order of preference.	\$puppet_enterprise::params::s
ssl_listen_port	Integer	Port that ACE runs on.	44633 (\$puppet_enterprise::ace_serv
allowlist	Array of strings	List of hosts that can connect to pe-ace-server.	[\$certname]

Related information

[Configure cipher suites](#) on page 223

Regulatory compliance or other security requirements might require you to change the cipher suites your SSL-enabled PE services use to communicate with other PE components.

Run Puppet on demand

You can use the orchestrator to run jobs from the console, the command line, or through the orchestrator API endpoints.

- [Run Puppet on demand from the console](#) on page 604

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 the targeted nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

- [Run Puppet on demand from the CLI](#) on page 611

Use the `puppet job run` command to start an on-demand Puppet run to enforce changes on your agent nodes.

Related information

[Orchestrator API v1](#) on page 678

You can use the orchestrator API to run jobs and plans on demand; schedule tasks and plans; get information about jobs, plans, and events; track node usage; and more.

Run 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 the targeted nodes in the appropriate order. Puppet compiles a new catalog for each node included in the job.

You can specify one, and only one, of these targets for a job:

- A list of one or more specific nodes.
- A node group.
- A Puppet Query Language (PQL) query defining a set of nodes.

Tip: When configuring the job in the console, if you switch the target method in the **Inventory** drop-down list, then the target list clears and you must re-select the target nodes. However, you can perform a one-time conversion of a PQL query to a static node list if you want to add or remove specific nodes from the query results.

Run Puppet on one or more specific nodes

An orchestrator job can target one or more specific nodes. This is useful if you want to run Puppet on a single node, a few specific nodes, nodes that are not in the same node group, or nodes that can't easily be identified by a single PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have the permissions necessary to run jobs.

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](#).

To schedule a recurring Puppet run or schedule a single run for a later time or date, refer to [Schedule a Puppet run](#) on page 609.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
4. 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. In the **Inventory** list, select **Node list**.
6. To create a node list, in the search field, start typing in the names of nodes to search for, and click **Search**. The search does not handle regular expressions.
7. Select the nodes you want to add to the job. You can select nodes from multiple searches to create a complete target node list.
8. Click **Run job**. View the job status and a list of previous Puppet jobs on the **Jobs** page. To rerun the job, click **Run again** and choose to rerun it on all nodes or only the nodes that failed during the initial run.

Run Puppet on a PQL query

An orchestrator job can target a set of nodes based on a PQL query. This is useful when you want to target a variable set of nodes that meet specific conditions, such as a particular operating system. When you supply a PQL query, the orchestrator runs the job on a list of nodes generated by the PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have the permissions necessary to run jobs and PQL queries.

To run PQL queries, you need the **View node data from PuppetDB** permission.

Tip: You can [Add custom PQL queries to the console](#) on page 235 to quickly select them when running jobs.

To schedule a recurring Puppet run or schedule a single run for a later time or date, refer to [Schedule a Puppet run](#) on page 609.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
4. 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. From the list of target types, select **PQL query**.

6. Specify a target by doing one of the following:

- Enter a query that selects the target you want. See the [Puppet Query Language \(PQL\) reference](#) 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	nodes[certname] { }
Nodes with a specific resource (example: httpd)	resources[certname] { type = "Service" and title = "httpd" }
Nodes with a specific fact and value (example: OS name is CentOS)	inventory[certname] { facts.os.name = "<OS>" }
Nodes with a specific report status (example: last run failed)	reports[certname] { latest_report_status = "failed" }
Nodes with a specific class (example: Apache)	resources[certname] { type = "Class" and title = "Apache" }
Nodes assigned to a specific environment (example: production)	nodes[certname] { catalog_environment = "production" }
Nodes with a specific version of a resource type (example: OpenSSL v1.1.0e)	resources[certname] { type = "Package" and title="openssl" and parameters.ensure = "1.0.1e-51.el7_2.7" }
Nodes with a specific resource and operating system (example: httpd and CentOS)	inventory[certname] { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }

7. Click **Submit query** and click **Refresh** to update the node results.

8. If you change or edit the query after it runs, click **Submit query** again.

9. Optional: To convert the PQL query target results to a node list, for use as a node list target, click **Convert query to static node list**.

Tip: If you select this option, the job target becomes a node list. You can add or remove nodes from the node list before running the job, but you cannot edit the query.

10. Click **Run job**.

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 the nodes that failed during the initial run.

Important: Unless you converted your PQL query to a node list, each time you run this job the PQL query runs again. Therefore, the job might run on a different set of nodes each time, depending on how your inventory has changed between runs. If you want the job to run on the same set of nodes queried when you originally created the query, you must convert the query to a node list before you run the job again.

Add custom PQL queries to the console

Add your own Puppet Query Language (PQL) queries to the console to quickly access them when running jobs.

For help forming queries, go to the PQL [Reference guide](#) in the Puppet documentation.

- On the primary server, copy the `custom_pql_queries.json.example` file, and remove the `.example` suffix. For example, you can use this command:

```
sudo cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

- Edit the file contents to include your own PQL queries or remove any existing queries.

- 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

An orchestrator job can target all nodes in a specific node group.

Before you begin

Make sure you have access to the nodes you want to target.

Note: If you don't have permission to view a node group, or the node group doesn't have any matching nodes, that node group isn't listed as an option. In addition, node groups don't appear if they have no rules specified.

To schedule a recurring Puppet run or schedule a single run for a later time or date, refer to [Schedule a Puppet run](#) on page 609.

- In the console, on the **Jobs** page, click **Run Puppet**.
- Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
- Select an environment:
 - Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
- 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.
- From the list of target types, select **Node group**.
- In the **Choose a node group** box, type or select a node group, and click **Select**.
- 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 the nodes that failed during the initial run.

Run jobs from other node lists

In addition to the **Jobs** page, you can run Puppet jobs on lists of nodes shown on the **Overview**, **Events**, and some **Node groups** pages.

Before you begin

Make sure you have the permissions necessary to run jobs.

For information about creating jobs on the **Jobs** page, refer to [Run Puppet on one or more specific nodes](#) on page 605, [Run Puppet on a PQL query](#) on page 605, and [Run Puppet on a node group](#) on page 608.

1. In the console, go to one of these pages and, depending on the page, go to the specific section or tab containing the node list you want to target:
 - **Overview:** This page shows a list of all your managed nodes, and gathers essential information about your infrastructure at a glance.
 - **Events page, Nodes with events section:** This page shows a summary of activity in your infrastructure and helps you analyze the details of important changes or investigate common causes behind related events. For example, if your Puppet runs are failing due to outdated code, after you update the code, you can create a job targeting the nodes listed as failed on the **Events** page. This ensures you're targeting the particular failed nodes you want to target.
 - **Node groups** pages for classification node groups: Node groups 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 those nodes at once.
2. Click **Run > Puppet**.

The list of nodes from the page, or page section, you were viewing is converted to a new Puppet run job list target.
3. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
4. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
5. Select the run mode for the job. The default run mode for a job always attempts to enforce new catalogs on nodes. To change the run mode, use the following selections:
 - **No-op:** Simulate a Puppet run on all nodes in the job without enforcing a new catalog.
 - **Debug:** Print all debugging messages.
 - **Trace:** Print stack traces on some errors.
 - **Eval-trace:** Display how long it took for each step to run.
 - **Override noop = true configuration:** If any nodes in the job have `noop = true` set in their `puppet.conf` files, Puppet ignores that setting and enforce a new catalog on those nodes. This setting corresponds to the `--no-noop` flag available on the orchestrator CLI.
6. **Important:** Do not change the **Inventory** from **Node list** to **PQL query**. This clears the node list target.
7. 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 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 or on a recurring schedule.

Before you begin

Make sure you have access to the nodes you want to target.

If a reboot occurs or you restore a backup, scheduled Puppet jobs are rescheduled based on the last execution time. If a reboot is caused by a scheduled Puppet job running in the orchestrator, that job returns a failed status.

1. In the console, on the **Jobs** page, click **Run Puppet**.
2. Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
3. Select an environment:
 - **Run nodes in their own assigned environment:** Nodes run in the environment specified by the Node Manager or their `puppet.conf` file.
 - **Select an environment for nodes to run in:** Select an environment from the list. Nodes can run in an environment if their environment is agent-specified or if they're classified in that environment by the node manager.
4. Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
5. Optional: To repeat the job on a regular schedule, change the run frequency from **Once** to **Hourly**, **Daily**, or **Weekly**.

Note: If a recurring job's run overlaps with the next scheduled run, the job skips the overlapped time and doesn't run again until the next scheduled start time.

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.

For details about the target options, refer to [Run Puppet on one or more specific nodes](#) on page 605, [Run Puppet on a PQL query](#) on page 605, and [Run Puppet on a node group](#) on page 608.

8. Click **Schedule job**.

Your job appears on the **Scheduled Puppet run** tab of the **Jobs** page.

Delete a scheduled job

Delete a job that you've scheduled to run at a later time or at recurring intervals.

If you want to delete a scheduled job created by another user, you must have the appropriate role-based permissions to do so.

1. In the console, go to **Jobs**.
2. Switch to the **Scheduled Puppet Run** tab.
3. Locate the scheduled job you want to delete and click **Remove**.

Stop an in-progress job

You can stop a job if, for example, you need to reconfigure a class or push a configuration change that the job needs. When you stop a Puppet job, in-progress jobs finish, and jobs that aren't started are canceled.

To stop a job:

- In the console, go to **Jobs**, switch to the **Puppet run** tab, locate the job you want to stop, and click **Stop**.

- If you started the job on the command line, press CTRL + C.

Run Puppet on demand from the CLI

Use the `puppet job run` command to start an on-demand Puppet run to enforce changes on your agent nodes.

Use the `puppet job run` command to immediately enforce change across nodes, rather than waiting for the next scheduled Puppet run. For example, if you add a new class parameter to a set of nodes, or if you deploy code to a new Puppet environment, you might want to use this command to run Puppet across all the nodes in the impacted environment.

Each time you use the `puppet job run` command, you can select one, and only one, of these targets:

- A list of one or more specific nodes, identified by certname.
- A node group, identified by node group ID.
- A Puppet Query Language (PQL) query defining a set of nodes.

The first time you run a command, you need to authenticate. For details, refer to [Setting PE RBAC permissions and token authentication for orchestrator](#) on page 601.

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 one or more specific nodes

An orchestrator job can target one or more specific nodes, identified by certname. This is useful if you want to run Puppet on a single node, a few specific nodes, nodes that are not in the same node group, or nodes that can't easily be identified by a PQL query.

Before you begin

You need to know the certnames of the node or nodes you want to target. Optionally, you can store the node names in a text file with each node name on a separate line.

Make sure you have the permissions necessary to run jobs.

Make sure you have access to the nodes you want to target.

1. Log into your primary server or the client tools workstation.

2. Run one of the following commands:

- To run a job on a single node run:

```
puppet job run --nodes <NODE_NAME>
```

- To run a job on multiple nodes, supply a comma-separated list of node names. Do not put spaces between the node names.

```
puppet job run --nodes <COMMA-SEPARATED_LIST_OF_NODE_NAMES>
```

- To run a job on a list of nodes in a text file, supply the path to the text file:

```
puppet job run --nodes @<PATH_TO_.txtFILE>
```

Tip: You can append additional options, such as `--noop`, after the node names or filepath. To learn about options you can supply to this command, refer to [puppet job run command options](#) on page 613.

When you execute the `puppet job run` command, the orchestrator generates a job ID for the job, shows you a list of nodes targeted by the job, and proceeds to run Puppet on the targeted nodes in the appropriate order. Puppet compiles a new catalog for all nodes targeted by the job.

To view the job status, run: `puppet job show <JOB_ID>`

To view a list of the 50 most-recent running and completed jobs, run: `puppet job show`

Run Puppet on a PQL query

An orchestrator job can target a set of nodes based on a PQL query. This is useful when you want to target a variable set of nodes that meet specific conditions, such as a particular operating system. When you supply a PQL query, the orchestrator runs the job on a list of nodes generated by the PQL query.

Before you begin

Make sure you have access to the nodes you want to target.

Make sure you have the permissions necessary to run jobs and PQL queries.

1. Log into your primary server or the client tools workstation.
2. Run one of the following commands:

- To supply the PQL query in the command:

```
puppet job run --query '<QUERY>'
```

For example:

```
puppet job run --query 'nodes[certname] { facts {name = "operatingsystem" and value = "Debian" } }'
```

- To supply the query in a text file:

```
puppet job run --query @<PATH_TO_.txt_FILE>
```

Tip: You can append additional options, such as `--noop`, after the query or filepath. To learn about options you can supply to this command, refer to [puppet job run command options](#) on page 613.

The following table shows some examples of PQL queries you might use for particular node targets.

Target	PQL query
A single node by certname	<code>'nodes { certname = "mynode" }'</code>
All nodes with <i>web</i> in their certname	<code>'nodes { certname ~ "web" }'</code>
All CentOS nodes	<code>'inventory { facts.os.name = "CentOS" }'</code>
All CentOS nodes with httpd managed	<code>'inventory { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }'</code>
All nodes with failed reports	<code>'reports { latest_report? = true and status = "failed" }'</code>
All nodes matching the environment of the last received catalog	<code>'nodes { catalog_environment = "production" }'</code>

Tip:

Make sure to wrap the entire query in single quotes and use double quotes inside the query.

To shorten the command, you can use `-q` in place of `--query`.

When you execute the `puppet job run` command, the orchestrator generates a job ID for the job, shows you a list of nodes targeted by the job, and proceeds to run Puppet on the targeted nodes in the appropriate order. Puppet compiles a new catalog for all nodes targeted by the job.

To view the job status, run: `puppet job show <JOB_ID>`

To view a list of the 50 most-recent running and completed jobs, run: `puppet job show`

Run Puppet on a node group

An orchestrator job can target all nodes in a specific node group, identified by node group ID.

Before you begin

You need to know the node group ID of the node group you want to target. You can use the node classifier API [GET /v1/groups](#) on page 516 endpoint to retrieve a list of node groups and their IDs.

Make sure you have the permissions necessary to run jobs.

Make sure you have access to the nodes you want to target.

1. Log into your primary server or the client tools workstation.
2. Run the following command:

```
puppet job run --node-group <NODE_GROUP_ID>
```

Tip: You can append additional options, such as `--noop`, after the node group ID. To learn about options you can supply to this command, refer to [puppet job run command options](#) on page 613.

When you execute the `puppet job run` command, the orchestrator generates a job ID for the job, shows you a list of nodes targeted by the job, and proceeds to run Puppet on the targeted nodes in the appropriate order. Puppet compiles a new catalog for all nodes targeted by the job.

To view the job status, run: `puppet job show <JOB_ID>`

To view a list of the 50 most-recent running and completed jobs, run: `puppet job show`

puppet job run command options

You might want to use these options with the `puppet job run` command. Alternately, use `puppet job --help` to get a complete list of options you can use with `puppet job` commands.

Option	Description
<code>--noop</code>	A flag indicating whether to run the job in no-op mode. No-op mode simulates changes from a new catalog without actually enforcing the changes. Excluding this option assumes <code>noop = false</code> , unless <code>noop</code> is specified elsewhere, such as the agent's <code>puppet.conf</code> file. Cannot be used in conjunction with <code>--no-noop</code> .
<code>--no-noop</code>	A flag indicating whether to run the job in enforcement mode and enforce a new catalog on all targeted nodes. This flag overrides <code>noop = true</code> if set in the agent's <code>puppet.conf</code> file. Cannot be used in conjunction with <code>--noop</code> .

Option	Description
--environment or -e	Supply an environment name, as a string, to override the environment specified in the orchestrator configuration file. The orchestrator uses this option to tell nodes which environment to run the job in. If any nodes can't run in the specified environment, those node runs fail. A node can run in an environment as long as it is classified into that environment in the PE node classifier.
--no-enforce-environment	A flag indicating whether you want the job to ignore the environment set by the --environment flag. When you use this flag, agents run in the environment specified by the PE Node Manager or their <code>puppet.conf</code> files.
--description	Supply a description of the job, as a string. The description appears on the job list and job details pages, and it is returned when you use the <code>puppet job show</code> command.
--concurrency	Supply an integer specifying the maximum number of nodes to run at one time. The default is an unlimited number of nodes. You can also configure concurrent compile requests in the console.

Post-run node status

After a Puppet run, the orchestrator returns a list of targeted nodes and their run statuses.

Node runs can be `in progress`, `completed`, `skipped`, or `failed`.

- For `completed` runs, 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 `in progress` runs, the orchestrator prints the elapsed time, in seconds, since the run started.
- When there is a `failed` run, subsequent or related runs might be `skipped`. For `failed` runs, the orchestrator prints an error message indicating why the run failed, a list of applications that were affected by the failure, and any applications that were affected by skipped node runs.

Use `puppet job show` to see a list of the 50 most-recent `in progress`, `completed`, and `failed` jobs.

In the console, on the **Jobs** page, you can review a list of jobs and to view job details for previous or in-progress jobs.

Stop an in-progress job

You can stop a job if, for example, you need to reconfigure a class or push a configuration change that the job needs. When you stop a Puppet job, in-progress jobs finish, and jobs that aren't started are canceled.

To stop a job:

- In the console, go to **Jobs**, switch to the **Puppet run** tab, locate the job you want to stop, and click **Stop**.
- If you started the job on the command line, press `CTRL + C`.

Tasks in PE

Tasks are ad-hoc actions you can execute on a target and run from the command line or the console.

A *task* is a single action that you execute on target machines. With tasks, you can troubleshoot and deploy changes to individual or multiple systems in your infrastructure.

You can run tasks from your tool of choice: the console, the orchestrator command line interface (CLI), or the orchestrator API [/command/task endpoint](#).

When you run a task, you can run it immediately, schedule it to run later, or schedule it to run at a recurring frequency - hourly, daily, weekly, every 2 weeks, or every four weeks. After you launch a task, you can check on the status or view the output later with the console or CLI.

If a reboot occurs or if you need to restore a backup, scheduled tasks are rescheduled based on the last execution time. If the scheduled task running in the orchestrator is what caused the reboot, the task run appears as failed.

Note: If you are running multiple tasks, make sure your `task_concurrency` and `bolt_server::concurrency` limits can accommodate your needs. To adjust these settings, go to [Orchestrator and pe-orchestration-services parameters](#) on page 236.

- [Installing tasks](#) on page 615

Puppet Enterprise comes with some pre-installed tasks, and you can install or write other tasks you want to use.

- [Running tasks in PE](#) on page 615

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 626

Bolt tasks are similar to scripts, but they are kept in modules and can have metadata. This allows you to reuse and share them.

Installing tasks

Puppet Enterprise comes with some pre-installed tasks, and you can install or write other tasks you want to use.

PE includes the following pre-installed tasks:

- `package` - Inspect, install, upgrade, and manage packages.
- `service` - Start, stop, restart, and check the status of a service.
- `facter_task` - Inspect the value of system facts.
- `puppet_conf` - Inspect Puppet agent configuration settings.

You can find other tasks packaged in Puppet modules that you can install from the [Forge](#) and then manage with a Puppetfile and Code Manager. To install a module (so that you can use the tasks in the module), select the desired install method under **Start using this module** on the module's Forge page and follow the presented instructions.

You can also [write tasks](#).

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

Running tasks in PE

Use the orchestrator to set up jobs in the console or on the command line and run Bolt tasks across systems in your infrastructure.

Running a task does not update your Puppet configuration. If you run a task that changes the state of a resource that Puppet is managing, a subsequent Puppet run changes the state of that resource back to what is defined in your Puppet configuration. For example, if you use a task to update the version of a managed package, the version of that package is reset to whatever is specified in a manifest on the next Puppet run.

Note: If you have set up compilers and you want to use tasks, you must either set `primary_uris` or `server_list` on agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.

- [Running tasks from the console](#) on page 616

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 622

Use the `puppet task run` command to run tasks on agent nodes.

- [Stop a task in progress](#) on page 626

You can stop a task that is currently running if, for example, you realize you need to adjust your PQL query or edit the task run parameters.

- [Inspecting tasks](#) on page 626

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 170

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.

- 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. You must express empty strings as two double-quotes with no space between (such as: " "). Structured values, like arrays, must be valid JSON.

Tasks that have default values use the default values when running unless you specify other values.

Note: The parameters you supply the first time you run a task are used for subsequent task runs when you use **Run again** on the **Task details** page.

- Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
- From the list of target types, select **Node list**.
- Create the node list.
 - Expand the **Inventory nodes** target.
 - Enter the name of the node you want to find and click **Search**.

Tip: The search does not handle regular expressions, but it does support partial matches.

- From the list of results, select the nodes that you want to add to your list. They are added to a table below.
- 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 check box 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 the nodes that failed during the initial run.

Tip: You can 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.

- In the console, in the **Orchestration** section, click **Tasks**.
- Click **Run a task** in the upper right corner of the **Tasks** page.
- In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to production.
- 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.

- Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

- 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. You must express empty strings as two double-quotes with no space between (such as: " "). Structured values, like arrays, must be valid JSON.

Tasks that have default values use the default values when running unless you specify other values.

Note: The parameters you supply the first time you run a task are used for subsequent task runs when you use **Run again** on the **Task details** page.

- Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
- From the list of target types, select **Node list**.
- Create the node list.

- Expand the **SSH nodes** target.

Restriction: This target is available only for tasks permitted to run on all nodes.

- Enter the target host names and the credentials required to access them. If you use an SSH key, include begin and end tags.
- 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**.
- Click **Add nodes**. They are added to the node table.
- 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 check box next to it and click **Remove selected**.

- 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 the nodes that failed during the initial run.

Tip: You can 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.

- In the console, in the **Orchestration** section, click **Tasks**.
- Click **Run a task** in the upper right corner of the **Tasks** page.
- In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.
- 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. You must express empty strings as two double-quotes with no space between (such as: " "). Structured values, like arrays, must be valid JSON.

Tasks that have default values use the default values when running unless you specify other values.

Note: The parameters you supply the first time you run a task are used for subsequent task runs when you use **Run again** 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.

Restriction: 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 the node table.
- 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 check box 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 the nodes that failed during the initial run.

Tip: You can 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 the permissions necessary 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**.

- 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.

- Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.
- 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. You must express empty strings as two double-quotes with no space between (such as: " "). Structured values, like arrays, must be valid JSON.

Tasks that have default values use the default values when running unless you specify other values.

Note: The parameters you supply the first time you run a task are used for subsequent task runs when you use **Run again** on the **Task details** page.

- Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
- From the list of target types, select **PQL query**.
- 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 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**.

Tip: 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 the nodes that failed during the initial run.

Tip: You can 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 Puppet Query Language (PQL) queries to the console to quickly access them when running jobs.

For help forming queries, go to the PQL [Reference guide](#) in the Puppet documentation.

- On the primary server, copy the `custom_pql_queries.json.example` file, and remove the `.example` suffix. For example, you can use this command:

```
sudo cp
/etc/puppetlabs/console-services/custom_pql_queries.json.example
/etc/puppetlabs/console-services/custom_pql_queries.json
```

- Edit the file contents to include your own PQL queries or remove any existing queries.

- 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 don't have permission to view a node group, or the node group doesn't have any matching nodes, that node group isn't listed as an option. In addition, node groups don't appear if they have no rules specified.

- In the console, in the **Orchestration** section, click **Tasks**.
- Click **Run a task** in the upper right corner of the **Tasks** page.
- In the **Code environment** field, select the environment where you installed the module containing the task you want to run. This defaults to **production**.
- 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.

- Optional: In the **Job description** field, provide a description. The text you enter here appears on the job list and job details pages.

- 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. You must express empty strings as two double-quotes with no space between (such as: " "). Structured values, like arrays, must be valid JSON.

Tasks that have default values use the default values when running unless you specify other values.

Note: The parameters you supply the first time you run a task are used for subsequent task runs when you use **Run again** on the **Task details** page.

- Optional: Under **Schedule**, select **Later** and choose a start date, time, time zone, and frequency for the job to run.
- From the list of target types, select **Node group**.
- In the **Choose a node group** box, type or select a node group, and click **Select**.
- 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 the nodes that failed during the initial run.

Tip: You can 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. For details, refer to [Setting PE RBAC permissions and token authentication for orchestrator](#) on page 601.

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 the permissions necessary 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>, <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 name=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 the permissions necessary 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 name=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 examples of PQL queries you might use for particular node targets.

Target	PQL query
A single node by certname	<code>'nodes { certname = "mynode" }'</code>
All nodes with <i>web</i> in their certname	<code>'nodes { certname ~ "web" }'</code>
All CentOS nodes	<code>'inventory { facts.os.name = "CentOS" }'</code>
All CentOS nodes with httpd managed	<code>'inventory { facts.operatingsystem = "CentOS" and resources { type = "Service" and title = "httpd" } }'</code>
All nodes with failed reports	<code>'reports { latest_report? = true and status = "failed" }'</code>

Target	PQL query
All nodes matching the environment of the last received catalog	'nodes { catalog_environment = "production" }'

Tip:

Make sure to wrap the entire query in single quotes and use double quotes inside the query.

To shorten the command, 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 516

Retrieves 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 name=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 that is currently running if, for example, you realize you need to adjust your PQL query or edit the task run parameters.

There are three ways to stop a task:

- In the PE console, go to the **Tasks** page, find the task run you want to stop, and click **Stop job**.
- On the command line, press **CTRL + C**.
- Use [POST /command/stop](#) on page 685.

When you stop a task, any Puppet runs that are already underway finish, but no new runs start on the node until you initiate the task again. While in-progress runs finish, the server continues to produce events for the job. The job's status changes to `stopped` once all in-progress runs finish.

Tip: If you need to stop in-progress Puppet runs (for example, if you need to stop a task that is hanging), use the `force` option with [POST /command/stop](#) on page 685.

Be aware that `force` immediately ends the job. This can result in an inconsistent or undesirable state due to job components (tasks, plans, Puppet runs, and so on) being ended prematurely.

If you can predict a scenario in which you'd want to force stop a task, we recommend getting familiar with [Forming orchestrator API requests](#) on page 678, specifically [POST /command/stop](#) on page 685 requests, so that you are prepared in the event you need to use this command.

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 17

Information is provided about the Puppet task specification that is compatible with Puppet Enterprise (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[^\\/\\\]*\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 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 implementations 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 637.

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.

Defining sensitive parameters

You can define task parameters, like passwords or API keys, as sensitive. The parameter is then masked when it appears in logs and API responses. When you want to view these values, set the log file to the correct level based on the service.

To define a parameter as sensitive within the JSON metadata, add the `"sensitive": true` property.

```

{
  "description": "This task has a sensitive property denoted by its
metadata",
  "input_method": "stdin",
  "parameters": {
    "user": {
      "description": "The user",
      "type": "String[1]"
    },
    "password": {
      "description": "The password",
      "type": "String[1]",
      "sensitive": true
    }
  }
}

```

Some services log sensitive parameter values. Here are the minimum log levels for each service where sensitive values can be seen.

Service	Minimum log level that shows sensitive parameters	Default log level
pe-bolt-server	INFO	INFO
pe-ace-server	INFO	INFO
pxp-agent	DEBUG	INFO
pcp-broker	TRACE	INFO

If you don't want sensitive parameters to be logged, configure the relevant service's log level to be one level higher than its minimum sensitive parameter value log level. For example, to avoid seeing sensitive parameters for pxp-agent, set the log level to INFO, which is one level higher than DEBUG, the minimum log level that shows sensitive parameters for pxp-agent.

Returning errors in tasks

To return a detailed error message if your task fails, include an `Error` object in the task's result.

Tip: When writing PowerShell or Bash scripts, it is a common practice to implement error handling that ignores errors during execution to allow the script to complete regardless of errors encountered. You might use variables such as `$ErrorActionPreference = SilenceContinue` or `set +e` for this. However, this can lead to a false sense of security by ignoring critical errors.

Don't use this type of error handling with Puppet tasks. By design, Puppet tasks are meant to highlight or stop upon encountering errors, with the intention that you'll address the discovered errors. Instead of ignoring errors, use the try-catch-finally technique. For more information refer to:

- [About Try Catch Finally](#) in the Microsoft PowerShell documentation.
- [Bash Basics: shell_try_catch.sh](#) on GitHub.

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)
```

Returning sensitive data

To return secrets from a task, use the `_sensitive` key in the output.

Here is an example of using the `_sensitive` key:

```
#!/opt/puppetlabs/puppet/bin/ruby
require 'json'

user_name = 'someone'
# Generate a 10 letter password
user_password = [*'a'...'z'].sample(10).join

result = { user: user_name, _sensitive: { password: user_password } }
```

```
puts result.to_json
```

When using the `_sensitive` key, PE treats the result as sensitive. The orchestrator redacts the value of the `_sensitive` key before storing it in the database.

Note: Redaction is a temporary solution for hiding the value of the `_sensitive` key. In a future release, PE will begin storing the value in an encrypted format in the database.

For a redaction example, given this output:

```
{"user": "foo_user", "_sensitive": {"password": "foo_password"}}
```

The orchestrator stores it in the database as follows:

```
{"user": "foo_user", "_sensitive": "Sensitive [value redacted]"}
```

The redacted value of the `_sensitive` key still appears in the following places:

- In the orchestrator API, in task run results.
- In the console, in task and plan run results.
- In logs that record the task output.

Note: The sensitive output is still written to the PXP agent spool directory and appears in the PXP agent logs for levels equal to or higher than the `debug` level.

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[1]]"
}

```

Define default parameters

You can define default task parameters, which are supplied to a task run even if the user does not specify a value for the parameter.

For example, the default location for this `log_location` parameter is `/var/log/puppetlabs`

```

"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[1]"
  },
  "password": {
    "description": "The password",
    "type": "String[1]",
    "sensitive": true
  }
}
}

```

Task metadata reference

The following table shows task metadata keys, values, and default values.

Metadata key	Description	Value	Default
description	A description of what the task does.	String	None
input_method	One or more input methods to use to pass parameters to the task.	environment, stdin, or powershell	For .ps1 tasks, the default value is powershell. For other tasks, the default is both environment and stdin.
parameters	The parameters or input types the task accepts, which must be valid Puppet types. You can provide an optional description. Refer to Task metadata types on page 644 and Adding parameters to metadata on page 642 for more information.	Array of objects describing each parameter	None (equivalent to Any)
puppet_task_version	The spec version used.	Integer	1 (which is the only valid value)
supports_noop	Whether the task supports no-op mode. Must be true for the task to accept the --noop option on the command line.	Boolean	false
implementations	A list of task implementations and the requirements used to select which one to run. Refer to Cross-platform tasks on page 629 for more information.	Array of objects describing each implementation	None

Metadata key	Description	Value	Default
files	A list of files to be provided when running the task, addressed by module. Refer to Sharing task code on page 631 for more information.	Array of strings	None

Task metadata types

Task metadata can accept most Puppet data types.

Restriction: Some Puppet types can not be represented as JSON, such as Hash[Integer, String], Object, or Resource. Do not use these in tasks, because they can never be matched.

Type	Description
String	Accepts any string.
String[1]	Accepts any non-empty string, which is a string at least one character in length.
Enum[choice1, choice2]	Accepts one of the listed choices.
Pattern[/\A\w+\z /]	Accepts strings matching the regex /\w+/ or non-empty strings of word characters.
Integer	Accepts integer values. JSON has no integer type, so this can vary depending on input.
Optional[String[1]]	This type designates the parameter as 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.

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 695 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 653 for information about RBAC considerations when writing plans or managing plan access.

Note:

If you have set up compilers and you want to use plans, you must set either `primary_uris` or `server_list` on your agents to point to your compilers.

If you are running multiple tasks or tasks within plans, make sure the `task_concurrency` and `bolt_server::concurrency` limits can support the number of tasks you need to run simultaneously. To adjust these settings, refer to [Orchestrator and pe-orchestration-services parameters](#) on page 236.

- [Plans in PE versus Bolt plans](#) on page 646

Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

- [Installing plans](#) on page 648

Plans are packaged in modules and deployed with Code Manager. PE includes some pre-installed plans. You can also download modules that contain plans from the Forge and write custom plans.

- [Running plans in PE](#) on page 649

The orchestrator can run plans across systems in your infrastructure. You can set up plan jobs from the Puppet Enterprise (PE) console or the command line. Plan jobs can run once or on a recurring schedule.

- [Writing plans](#) on page 652

Plans allow you to run more than one task with a single command, compute values for the input to a task, and run other complex workflows at the same time. They also allow you greater flexibility for creating custom RBAC limitations by wrapping tasks and other workflows within plans.

Plans in PE versus Bolt plans

Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

Unavailable plan language functions

The following Bolt plan functions don't work in PE because they haven't been implemented yet or cause issues during plan runs:

- `add_to_group`
- `background`
- `dir::children`
- `download_file`
- `file::exists`
- `file::readable`
- `file::write`
- `get_resources`
- `out::verbose`
- `parallelize`
- `prompt`
- `prompt::menu`
- `remove_from_group`
- `resolve_references`
- `resource`
- `run_task_with`
- `set_config`
- `set_feature`
- `set_resources`
- `set_var`
- `system::env`
- `wait`
- `write_file`

Apply blocks

The apply feature, including `apply_prep`, only works for targets using the PXP agent and the PCP transport. It fails on remote devices and on targets connected via SSH or WinRM.

Target groups

Support for target groups is unavailable in PE. Using `add_to_group` causes a plan to fail and referencing a group name in `get_targets` doesn't return any nodes. When using `get_targets` you must reference either node certnames or supply a PuppetDB query. Here is an example of a plan using `get_targets` with node certnames:

```
plan example::get_targets_example () {
  $nodes = get_targets(['node1.example.com', 'node2.example.com'])
  run_command('whoami', $nodes)
}
```

Target behaviors

PE assumes all target references can be matched to one of the following:

- A connected agent with a certname
- An entry in the PE inventory service

Therefore, target names must match either a certname or an entry in the PE inventory service.

New targets can't be added to the inventory service inside a plan. New target objects created in plans can't connect because PE can'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 can'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 `file::read`, the source location for the files **must** be from a module that uses a `modulename/filename` selector for a file or directory in `$MODULEROOT/files`. PE does not support file sources that reference absolute paths.

The following two code examples show a module structure and a plan that correctly use 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

Using plans in PE requires [Managing code with Code Manager](#) on page 774. You must enable Code Manager to deploy code to your primary server.

Primary servers deploy a second `codedir` from which plans load code. This secondary code location on your primary server impacts standard module functionality:

- You can't use the `puppet module install` command to install modules for plans, because the `puppet module` tool can't install to the plan `codedir`. However, the `puppet module install` command works as usual for non-plan Puppet code executed and compiled from Puppet Server.
- A `$modulepath` configuration that uses fully qualified paths might not work for plans if they reference the standard `/etc/puppetlabs/code` location. It is more reliable to use relative paths in `$modulepath`.

Installing plans

Plans are packaged in modules and deployed with Code Manager. PE includes some pre-installed plans. You can also download modules that contain plans from the Forge and write custom plans.

Important: Built-in plans can work without Code Manager; however, you must use Code Manager to install custom plans and plans from the Forge.

To install a new module containing a plan:

1. Find the module you want on the Forge.
2. Under **Start using this module**, select **r10k** or **Code Manager** as the **Installation method**.
3. Using Code Manager, follow the instructions. You must use Code Manager.

In the Forge, modules containing plans have a **Plans** section in the README.

You can also learn how to [write plans in Puppet language](#) and check out some [example plans](#).

Running plans in PE

The orchestrator can run plans across systems in your infrastructure. You can set up plan jobs from the Puppet Enterprise (PE) console or the command line. Plan jobs can run once or on a recurring schedule.

Plans can't change your Puppet configuration, which defines the desired state of your infrastructure. If you run a plan that changes the state of a resource managed by Puppet (for example, upgrading a package or service), the next Puppet run changes that resource's state back to what is defined in your Puppet configuration.

If you have set up compilers and you want to use plans, you must set either `primary_uris` or `server_list` on your agents to point to your compilers. This setting is described in the section on configuring compilers for orchestrator scale.

Important:

Plans in PE require [Managing code with Code Manager](#) on page 774.

Unlike [Tasks in PE](#) on page 614, you cannot stop a plan that is in progress.

- [Running plans from the console](#) on page 649

Run ad hoc plans from the console.

- [Running plans from the command line](#) on page 650

Run a plan using the `puppet plan run` command.

- [Inspecting plans](#) on page 651

View the plans you have installed and which ones you have permissions to run, as well as individual plan metadata.

- [Running plans alongside code deployments](#) on page 651

The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

Running plans from the console

Run ad hoc plans from the console.

Before you begin

Install the tasks you want to use.

Make sure you have permission to run the tasks.

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 job 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 651

- Credentials, if required, formatted with the `--user` and `--password` flags.

For example, if a plan defined in `mymodule/plans/myplan.pp` accepts a `load_balancer` parameter, run:

```
puppet plan run mymodule::myplan load_balancer=lb.myorg.com
```

You can pass a comma-separated list of node names, wildcard patterns, or group IDs to a plan parameter that is passed to a run function or that the plan resolves using `get_targets`.

Related information

[GET /v1/groups](#) on page 516

Retrieves a list of all node groups in the node classifier.

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.

Running plans alongside code deployments

The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

Before you begin

Plans in Puppet Enterprise (PE) require [Managing code with Code Manager](#) on page 774.

You might want to disable the file sync locking mechanism if:

- You want to allow code deployments to complete while plans are running.
- You want to allow plans to start while code deployments are in progress.
- Your code deployments don't frequently or substantially change the environment state that plans run in.
- You aren't concerned if the environment state changes (due to a concurrent code deployment) during a plan run.



CAUTION:

When you disable the file sync locking mechanism, the environment states your plans run in might be inconsistent or change while the plans are starting, running, or finishing. This depends on when your code deployments happen and whether they happen while a plan is running. Puppet functions and plans that call other plans might behave unexpectedly if a code deployment occurs while a plan is running.

If it is important to you that your plans **always** run in a consistent environment state, you probably **don't** want to disable the file sync locking mechanism.

1. In the PE console, go to **Node groups > PE Infrastructure > PE Orchestrator**.
2. On the **Classes** tab, locate (or add) the `puppet_enterprise::profile::plan_executor` class, and set the `versioned_deploys` parameter to `true`. The full declaration is:

```
puppet_enterprise::profile::plan_executor::versioned_deploys: true
```

Important: Setting this parameter to `true` **disables** the file sync client's locking mechanism that usually enforces a consistent environment state for your plans. The locking mechanism prevents plans from starting while a code deployment occurs and forces code deployments to wait while a plan is in progress.

Tasks, scripts, and apply block compilations always use the latest synced version of your code, regardless of this setting. However, after you set `versioned_deploys` to `true`, Puppet functions and plans that call other plans also use the latest synced version of your code, instead of the version of the code that was present when the plan started. Due to the possibility for the code to change during the plan run, Puppet functions and plans that call other plans might behave unexpectedly if a code deployment occurs while a plan is running.

If you want to enforce a consistent environment state for plans, set `versioned_deploys` to `false`. If [Code deployments time out](#) on page 806 while waiting for long-running plans to finish, adjust the `timeouts_sync` setting in your [Code Manager parameters](#) on page 791.

3. Commit your changes.
4. The orchestrator server **doesn't** automatically restart after setting this parameter, so you must restart the `pe-orchestration-services` service to finish applying the change. To do this, run the following command on the primary server:

```
service pe-orchestration-services reload
```

Plans and code deployments now start and finish without blocking each other.

Related information

[Declare classes](#) on page 445

Classes are blocks of Puppet code that configure nodes and assign resources to nodes.

[Lockless code deploys](#) on page 783

The *lockless code deploys* feature within Code Manager allows deployment of Puppet code without interrupting other Puppet operations. When this feature is disabled, requests to Puppet Server are blocked during code deployments until the file sync client has finished updating the live Puppet code directory. However, when lockless code deploys are enabled, the file sync client saves newly deployed code into versioned directories, ensuring that the live code directory is not overwritten. This process allows Puppet operations to continue without interruption during code deployments.

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 653

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 665

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 `1:2.17.0-1ubuntu1` and `1:2.17.1-1ubuntu0.4` only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```

You can also use PuppetDB queries to select parameter types.

For example, if you need to restrict the targets that `infra::upgrade_git` can run on, use a PuppetDB query to identify which targets are selected for the `git` upgrade.

```
plan infra::upgrade_git (
  Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
  # Use puppetdb to find the nodes from the "other" team's web cluster
  $query = [from, nodes, [=, [fact, cluster], "other_team"]]
  $selected_nodes = puppetdb_query($query).map() |$target| {
    $target[certname]
  }
  run_task('package', $selected_nodes, name => 'git', action => 'upgrade', version => $version)
}
```

Specifying plan parameters

Specify plan parameters to do things like determine which targets to run different parts of your plan on. You can pass a parameter as a single target name, comma-separated list of target names, Target data type, or array. The target names can be either certnames or inventory node names.

The example plan below shows the target parameters `$load_balancers` and `$webservers` specified as data type `TargetSpec`. The plan then calls the `run_task` function to specify which targets to run the tasks on. The Target names are collected and stored in `$webserver_names` by iterating over the list of Target objects returned by `get_targets`. Task parameters are serialized to JSON format so that extracting the names into an array of strings ensures that the `webservers` parameter is in a format that can be converted to JSON.

```
plan mymodule::my_plan(
  TargetSpec $load_balancer,
  TargetSpec $webservers,
) {

  # Extract the Target name from $webservers
  $webserver_names = get_targets($webservers).map |$n| { $n.name }

  # process webservers
  run_task('mymodule::lb_remove', $load_balancer, webservers => $webserver_names)
  run_task('mymodule::update_frontend_app', $webservers, version => '1.2.3')
  run_task('mymodule::lb_add', $load_balancer, webservers => $webserver_names)
}
```

To execute this plan from the command line, pass the parameters as `parameter=value`. The `TargetsSpec` accepts either an array as JSON or a comma separated string of target names.

```
puppet plan run mymodule::myplan
load_balancer=lb.myorg.com
webservers='["kermit.myorg.com", "gonzo.myorg.com"]'
```

Alternatively, here is an example of the same plan, run on the same targets, using the Orchestrator API [POST /command/plan_run](#) on page 695 endpoint:

```
curl -k -X POST -H "Content-Type: application/json" \
-H "X-Authentication:$TOKEN" \
-d '{ "environment": "$ENV", "plan_name": "mymodule::myplan", \
"params": { "targets": "$TARGET_NAME", "load_balancer": "lb.myorg.com", \
"webservers": [ "kermit.myorg.com", "gonzo.myorg.com" ] } }' \
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"
```

Parameters that are passed to the `run_*` plan functions are serialized to JSON. For example, in the plan below, the default value of `$example_nul` is `undef`. The plan calls the `test::demo_undef_bash` with the `example_nul` parameter.

```
plan test::parameter_passing (
  TargetSpec $targets,
  Optional[String[1]] $example_nul = undef,
) {
  return run_task('test::demo_undef_bash', $targets, example_nul =>
$example_nul)
}
```

The implementation of the `demo_undef_bash.sh` task is:

```
#!/bin/bash
example_env=$PT_example_nul
echo "Environment: $PT_example_nul"
echo "Stdin:"
cat -
```

By default, the task expects parameters passed as a JSON string on `stdin` to be accessible in prefixed environment variables.

Additionally, you can use the Orchestrator API [POST /command/plan_run](#) on page 695 endpoint with token authentication, such as:

```
curl -k -X POST -H "Content-Type: application/json" -H
 "X-Authentication:$TOKEN" -d '{ "environment": "$ENV",
"plan_name": "test::parameter_passing", "params": { "targets": \
"$TARGET_NAME" } }' \
"https://$PRIMARY_HOST:8143/orchestrator/v1/command/plan_run"
```

Using Hiera data in plans

Use the `lookup()` function in plans to look up Hiera data. You can look up data inside or outside of `apply` blocks, or use the `plan_hierarchy` key to look up data both inside and outside `apply` blocks within the same plan.

Inside [apply blocks](#), PE compiles catalogs for each target and has unlimited access to your Hiera data. You can use the same Hiera configuration, data, and lookup process as you do throughout PE.

Outside `apply` blocks, the plan executes a script, doesn't have a concept of a target or context, and cannot load per-target data. These limitations make some common Hiera features, like interpolating target facts, incompatible with plans in PE outside of `apply` blocks.

You can look up static Hiera data outside of apply blocks by adding a `plan_hierarchy` key to your Hiera configuration at the same level as the `hierarchy` key. This allows you to look up data inside and outside apply blocks in the same plan, enabling you to use your existing Hiera configuration in plans without encountering an error if per-target interpolations exist and your plan tries to look up data outside an apply block.

Static Hiera data is also useful for user-specific data that you want the plan to look up.

For example, consider the Hiera configuration below at `<ENV_DIR>/hiera.yaml`.

```
version: 5
hierarchy:
  - name: "Target specific data"
    path: "targets/%{trusted.certname}.yaml"
  - name: "Per-OS defaults"
    path: "os/%{facts.os.family}.yaml"
  - name: Common
    path: hierarchy.yaml

plan_hierarchy:
  - name: Common
    path: plan_hierarchy.yaml
```

You can set a user-specific API key in the `plan_hierarchy.yaml` data file, as well as use Hiera to look up a per-target filepath inside an apply block by using the following pieces of data:

Use the following data located at `<ENV_DIR>/data/plan_hierarchy.yaml`:

```
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'] == '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'] == 'CentOS' }
  $ubuntu_targets = get_targets($targets).filter |$n| { $n.facts['os'] == '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.
<code>filter_set(block)</code>	Filters a <code>ResultSet</code> with the given block and returns a <code>ResultSet</code> object (where the <code>filter</code> 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 <code>Boolean</code> 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>	true	Returns a <code>ResultSet</code> , including failed results, rather than failing the plan. Boolean.
<code>_description</code>	none	Adds a description to the <code>apply</code> block. String.
<code>_noop</code>	true	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_logging()` 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 `1:2.17.0-1ubuntu1` and `1:2.17.1-1ubuntu0.4` only.

```
plan infra::upgrade_git (
  TargetSpec $targets,
  Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
  run_task('package', $targets, name => 'git', action => 'upgrade', version => $version)
}
```

You can also use PuppetDB queries to select parameter types.

For example, if you need to restrict the targets that `infra::upgrade_git` can run on, use a PuppetDB query to identify which targets are selected for the git upgrade.

```
plan infra::upgrade_git (
    Enum['1:2.17.0-1ubuntu1', '1:2.17.1-1ubuntu0.4'] $version,
) {
    # Use puppetdb to find the nodes from the "other" team's web cluster
    $query = [from, nodes, [=, [fact, cluster], "other_team"]]
    $selected_nodes = puppetdb_query($query).map() |$target| {
        $target[certname]
    }
    run_task('package', $selected_nodes, name => 'git', action => 'upgrade',
    version => $version)
}
```

Plan structure

YAML plans contain a list of steps with optional parameters and results.

YAML maps accept these keys:

- `steps`: The list of steps to perform
- `parameters`: (Optional) The parameters accepted by the plan
- `return`: (Optional) The value to return from the plan

Steps key

The `steps` key is an array of step objects, each of which corresponds to a specific action to take.

When the plan runs, each step is executed in order. If a step fails, the plan halts execution and raises an error containing the result of the step that failed.

Steps use these fields:

- `name`: A unique name that can be used to refer to the result of the step later
- `description`: (Optional) An explanation of what the step is doing

Other available keys depend on the type of step.

Command step

Use a command step to run a single command on a list of targets and save the results, containing `stdout`, `stderr`, and `exit code`.

The step fails if the exit code of any command is non-zero.

Command steps use these fields:

- `command`: The command to run
- `target`: A target or list of targets to run the command on

For example:

```
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 webservers using the built-in facts
    plan"
    parameters:
      nodes:
        - web1.example.com
        - web2.example.com
        - web3.example.com

```

Resources step

Use a `resources` step to apply a list of Puppet resources. A resource defines the desired state for part of a target. Bolt ensures each resource is in its desired state. Like the steps in a `plan`, if any resource in the list fails, the rest are skipped.

For each `resources` step, Bolt executes the `apply_prep` plan function against the targets specified with the `targets` field. For more information about `apply_prep` see the Applying manifest block section.

Resources steps use these fields:

- `resources`: An array of resources to apply
- `target`: A target or list of targets to apply the resources on

Each resource is a YAML map with a type and title, and optionally a `parameters` key. The resource type and title can either be specified separately with the `type` and `title` keys, or can be specified in a single line by using the type name as a key with the title as its value.

For example:

```

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 web servers"
```

Parameters key

Plans accept parameters in 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, which are all optional:

- `type`: Specify a valid [Puppet data type](#). The plan fails if the value supplied to the parameter does not match the defined type. If you do not specify a data type, this field defaults to `Any`.
- `default`: Specify a default value to use if no specific value is supplied to the parameter. This can be empty.
- `description`: A description of the parameter. This can be empty.

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: web1.example.com
    parameters:
      message: "The count is ${count}, and twice the count is
${double_count}"
```

Returning results

You can return a result from a plan by setting the `return` key at the top level of the plan. When the plan finishes, the `return` key is evaluated and returned as the result of the plan. If no `return` key is set, the plan returns `undef`

```
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 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')} ")
  }
}
```

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.

Orchestrator API v1

You can use the orchestrator API to run jobs and plans on demand; schedule tasks and plans; get information about jobs, plans, and events; track node usage; and more.

- [Forming orchestrator API requests](#) on page 678

The orchestrator API accepts well-formed HTTPS requests and requires authentication.

- [Root endpoints](#) on page 679

Use the `orchestrator` endpoint to get orchestrator API metadata.

- [Command endpoints](#) on page 681

Use the command endpoints to run Puppet, jobs, and plans on demand or stop in-progress jobs. You can also create task-targets, which provide privilege escalation for users who would otherwise not be able to run certain tasks or run tasks on certain nodes or node groups.

- [Inventory endpoints](#) on page 700

Use the `inventory` endpoints to check whether the orchestrator can reach a node.

- [Jobs endpoints](#) on page 703

Use the `jobs` endpoints to examine jobs and their details.

- [Scheduled jobs endpoints](#) on page 717

Use the `scheduled_jobs` endpoints to query, edit, and delete scheduled orchestrator jobs.

- [Plans endpoints](#) on page 732

Use the `plans` endpoints to get information about plans.

- [Plan jobs endpoints](#) on page 736

Use the `plan_jobs` endpoints to examine plan jobs and their details.

- [Tasks endpoints](#) on page 749

Use the `tasks` endpoints to get information about tasks you've installed and tasks included with Puppet Enterprise (PE).

- [Usage endpoints](#) on page 753

Use the `usage` endpoint to view details about your deployment's active nodes.

- [Scopes endpoints](#) on page 755

Use the `scopes` endpoints to retrieve information about task-targets.

- [Orchestrator API error responses](#) on page 758

Orchestrator API error responses are formatted as JSON objects.

Related information

[API index](#) on page 33

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

Forming orchestrator API requests

The orchestrator API accepts well-formed HTTPS requests and requires authentication.

Orchestrator API requests must include a URI path following the pattern:

```
https://<DNS>:8143/orchestrator/v1/<ENDPOINT>
```

The variable path components derive from:

- DNS: Your PE console host's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet` command (as explained in [Using example commands](#) on page 28).
- ENDPOINT: One or more sections specifying the endpoint, such as `command` or `jobs`. Some endpoints require additional sections, such as [POST /command/deploy](#) on page 681.

For example, you could use any of these paths to call the [GET /inventory](#) on page 700 endpoint:

```
https://$(puppet config print server):8143/orchestrator/v1/inventory
https://localhost:8143/orchestrator/v1/inventory
```

```
https://puppet.example.dns:8143/orchestrator/v1/inventory
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Orchestrator API authentication

You must authenticate orchestrator API requests with user authentication tokens. For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Token-based authentication](#) on page 303.

To use a token in an orchestrator API request, you can use `puppet-access show`, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs"

curl --insecure --header "$auth_header" "$uri"
```

Or you can use the actual token, such as:

```
auth_header="X-Authentication: <TOKEN>
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs

curl --insecure --header "$auth_header" "$uri"
```

Related information

[Token-based authentication](#) on page 303

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 role-based access control (RBAC), and they provide the user with the appropriate access to HTTP requests.

Root endpoints

Use the `orchestrator` endpoint to get orchestrator API metadata.

GET /orchestrator

Returns metadata about the orchestrator API, along with a list of links to application management resources.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the request is a basic call, such as:

```
GET https://orchestrator.example.com:8143/orchestrator
```

The GET `/orchestrator` endpoint does not support any parameters; however, as with other orchestrator API endpoints, you must provide authentication.

Response format

The response is a JSON object using these keys:

Key	Definition
info	Contains the API title, description, version compatibility warnings, the current version, and license information.

Key	Definition
<code>status</code>	A URI path you can call to check the orchestrator API status. To check the status of orchestrator services, use the Status API on page 416.
<code>collections</code>	<p>URI paths you can use to call various endpoints, such as the Jobs endpoints on page 703 and the <code>environments</code> endpoint.</p> <p>The <code>environments</code> endpoint response tells you either which environments are available or whether a named environment exists.</p>
<code>commands</code>	URI paths for the Command endpoints on page 681.

For example:

```
{
  "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

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 500 response if the orchestrator API can't be reached.

Command endpoints

Use the command endpoints to run Puppet, jobs, and plans on demand or stop in-progress jobs. You can also create task-targets, which provide privilege escalation for users who would otherwise not be able to run certain tasks or run tasks on certain nodes or node groups.

You can:

- [POST /command/deploy](#) on page 681: Run Puppet on demand.
- [POST /command/stop](#) on page 685: Stop an orchestrator job that is currently running.
- [POST /command/stop_plan](#) on page 686: Stop an orchestrator plan job that is currently running.
- [POST /command/task](#) on page 687: Run a task on a set of nodes.
- [POST /command/task_target](#) on page 691: Define a set of tasks and nodes/node groups you can use to escalate privileges for users who would otherwise not be able to run those tasks or run tasks on those nodes or node groups.
- [POST /command/plan_run](#) on page 695: Run a plan.
- [POST /command/environment_plan_run](#) on page 696: Run a plan in a specified environment.

The `schedule_deploy`, `schedule_task`, and `schedule_plan` endpoints are deprecated. Instead, use [POST /scheduled_jobs/environment_jobs](#) on page 724.

POST /command/deploy

Run Puppet on demand – run the orchestrator across all nodes in an environment.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object using keys described in the following table. The `environment` key is always required. Additional keys might be required depending on the values of other keys.

Key	Format	Definition
<code>environment</code>	String	Required: The name of the environment to deploy or an empty string. If you supply an empty string, you must set <code>enforce_environment</code> to <code>false</code> .

Key	Format	Definition
scope	JSON object	<p>Contains exactly one key defining the deployment target:</p> <ul style="list-style-type: none"> • <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>: The ID of a classifier node group that has defined rules. The node group itself must have defined rules – It is not sufficient for only the node group's parent groups to define rules. The user submitting the request must also have permissions to view the specified node group.
		Required if <code>environment</code> is an empty string.
concurrency	Integer or range	<p>The maximum number of nodes to run at one time. The default is a range between 1 and the value of the <code>global_concurrent_compiles</code> parameter.</p>
		<p>For information about <code>global_concurrent_compiles</code>, refer to Orchestrator and pe-orchestration-services parameters on page 236.</p>
debug	Boolean	<p>Whether to use the <code>--debug</code> flag on Puppet agent runs.</p>
description	String	<p>A description of the job.</p>
enforce_environment	Boolean	<p>Whether to force agents to run in the specified environment. This key must be <code>false</code> if <code>environment</code> is an empty string.</p>
evaltrace	Boolean	<p>Whether to use the <code>--evaltrace</code> flag on Puppet agent runs.</p>

Key	Format	Definition
filetimeout	Integer	The value for the --filetimeout flag on Puppet agent runs.
http_connect_timeout	Integer	The value for the --http_connect_timeout flag on Puppet agent runs.
http_keepalive_timeout	Integer	The value for the --http_keepalive_timeout flag on Puppet agent runs.
http_read_timeout	Integer	The value for the --http_read_timeout flag on Puppet agent runs.
noop	Boolean	Whether to run the agent in no-op mode. The default is false.
no_noop	Boolean	Whether to run the agent in enforcement mode. The default is false. This flag overrides noop = true if set in the agent's puppet.conf file. This flag can't be set to true if the noop flag is also set to true.
ordering	String	Sets the --ordering flag on Puppet agent runs.
skip_tags	String	Sets the --skip_tags flag on Puppet agent runs.
tags	String	Sets the --tags flag on Puppet agent runs.
trace	Boolean	Whether to use the --trace flag on Puppet agent runs.
use_cached_catalog	Boolean	Whether to use the --use_cached_catalog flag on Puppet agent runs.
usecacheonfailure	Boolean	Whether to use the --usecacheonfailure flag on Puppet agent runs.
userdata	JSON object	Arbitrary key/value data supplied to the job.

Here is an example of a complete curl command for the /command/deploy endpoint. This request targets nodes in the All Nodes node group in the production environment:

```

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"

```

The following are additional examples of valid JSON bodies for the /command/deploy endpoint.

This body deploys two specific nodes in the production environment:

```
{
  "environment" : "production",
  "scope" : {
    "nodes" : [ "node1.example.com", "node2.example.com" ]
  }
}
```

This body deploys the node1.example.com node in no-op mode:

```
{
  "environment" : "",
  "enforce_environment": false,
  "noop" : true,
  "scope" : {
    "nodes" : [ "node1.example.com" ]
  },
  "userdata": {
    "servicenow_ticket": "INC0011211"
  }
}
```

This body deploys any node in the production environment with a certname value matching a regex:

```
{
  "environment" : "production",
  "scope" : {
    "query" : [ "from", "nodes", [ "~", "certname", ".*" ] ]
  }
}
```

Response format

If all node runs succeed and the environment is successfully deployed, the server returns 202 and a JSON object using these keys:

- **id**: An absolute URL that links to the newly created job.
- **name**: A stringified number identifying the newly created job. You can use this with other endpoints, such as [GET /jobs/<job-id>](#) on page 708 (retrieve information about the status of the job) and [POST /command/stop](#) on page 685.

For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/81"
    "name" : "81"
  }
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Value	Definition
404	puppetlabs/orchestrator/unknown-environment	The specified environment does not exist.

Response code	Value	Definition
400	puppetlabs/orchestrator/empty-environment	The specified environment contains no applications or no nodes.
400	puppetlabs/orchestrator/empty-target	The specified scope resolves to an empty list of nodes.
400	puppetlabs/orchestrator/puppetdb-error	If the request specified a query for the scope, the orchestrator is unable to make a query to PuppetDB.
400	puppetlabs/orchestrator/query-error	If the request specified a query for the scope, the query is invalid or the user submitting the request does not have permission to run the query.

POST /command/stop

Stop an orchestrator job that is currently in progress.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object containing the `job` key, which specifies the job ID of the job to stop, such as:

```
{
  "job": "1234"
}
```

Job IDs are returned in responses from [POST /command/deploy](#) on page 681 and [GET /jobs](#) on page 703.

By default, this request halts the specified job. This prevents the job from starting new Puppet agent runs but allows any in-progress runs to finish. While in-progress runs are finishing, the server continues to produce events for the job. The job's status changes to `stopped` once all in-progress runs finish.

Tip: If you want to completely stop the job (to stop in-progress runs *and* prevent new runs from starting), add the `force` key to the request, such as:

```
{
  "job": "1234"
  "force": true
}
```

You can `force`, for example, to stop a task that is hanging. Be aware that `force` immediately ends the job. This can result in an inconsistent or undesirable state due to job components (tasks, plans, Puppet runs, and so on) being ended prematurely.

Here is an example of a complete curl command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/
stop"
data='{"job": "1234"}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

`POST /command/stop` is *idempotent* – you can use it against the same job any number of times.

Response format

If the job is stopped successfully, the server returns a 202 response and a JSON object containing these keys:

- `id`: An absolute URL that links to the stopped job. This is based on the `job` key in the request.
- `name`: A stringified number identifying the stopped job.
- `nodes`: A hash showing all possible node statuses, and how many nodes are currently in each 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
    }
  }
}
```

Important: When a job is successfully stopped, any in-progress Puppet agent runs finish, but no new agent runs start. While agents are finishing, the server continues to produce events for the job, and the job itself transitions to stopped status when all in-progress agent runs have finished.

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the `kind` key:

Response code	Key	Definition
400	puppetlabs/orchestrator/validation-error	The specified job ID is not formatted correctly or is otherwise not valid.
404	puppetlabs/orchestrator/unknown-job	The specified job ID does not exist.

POST /command/stop_plan

Stop an orchestrator plan job that is currently in progress.

This command interrupts the thread that is running the plan.

If the plan doesn't have code to explicitly handle the interrupt, the plan finishes with an error. If the plan can handle the interrupt, whether or not the plan stops depends on the plan's interruption handling.

If the plan is running a task (or otherwise) when interrupted, an error occurs and the plan stops, but the underlying in-progress task job finishes. If you need to force stop an in-progress job, use [POST /command/stop](#) on page 685.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object containing the `plan_job` key, which specifies the string-formatted ID of the plan job to stop, such as:

```
{
  "plan_job" : "1234"
```

```
}
```

The `plan_job` ID is the name value that is returned from [POST /command/plan_run](#) on page 695 and [GET /plan_jobs](#) on page 736.

Here is an example of a complete curl command:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/
stop_plan"
data='{"plan_job": "1234"}'

curl --insecure --header "$type_header" --header "$auth_header" -X POST
"$uri" --data "$data"
```

`POST /command/stop_plan` is *idempotent* – you can use it against the same plan job any number of times.

Response format

If the specified plan job exists, the server returns a 202 response containing the `name` key, which is the same as the `plan_job` key in the request. For example:

```
{
  "name": "1234"
}
```

Important:

If the plan is running a task (or otherwise) when interrupted, the plan stops, but the underlying in-progress task job finishes. If you need to force stop an in-progress job, use [POST /command/stop](#) on page 685.

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the `kind` key:

Response code	Key	Definition
400	puppetlabs/orchestrator/validation-error	The specified plan job ID is not formatted correctly or is otherwise not valid.
404	puppetlabs/orchestrator/unknown-job	The specified plan job ID does not exist.

POST /command/task

Run a task on a set of nodes. The task does not run on any nodes in the defined scope that you do not have permission to run tasks on.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object using keys described in the following table. Most keys are required, some keys are optional, and some required keys can be empty.

Key	Format	Definition
environment	String	Required: The name of the environment to load the task from. The default is <code>production</code> .
scope	JSON object	Required: Contains exactly one key defining the nodes to run the task on: <ul style="list-style-type: none"> <code>nodes</code>: An array 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>: The ID of a classifier node group that has defined rules. The node group itself must have defined rules – It is not sufficient for only the node group's parent groups to define rules. The user submitting the request must also have permissions to view the specified node group. The task does not run on any nodes specified in the scope that the user does not have permission to run the task on.
description	String	A optional description of the job.
noop	Boolean	Whether to run the job in no-op mode. The default is <code>false</code> .
params	JSON object	Required: Parameters to pass to the task. Can be an empty object.
targets	Array of JSON objects	Required: A collection of keys used to run the task on nodes through SSH or WinRM via Bolt server, such as user account information, <code>runcwd</code> specifications, or a designated temporary directory. Refer to the Targets section, below, for information about optional and required keys to use in <code>targets</code> .

Key	Format	Definition
task	String	Required: The task to run on the target nodes. Use the GET /tasks on page 749 endpoint to get task names.
userdata	JSON object	Optional arbitrary key/value data supplied to the job.

For example, this body runs the package task on the node1.example.com node in the test-env-1 environment. It passes action and name parameters to the task.

```
{
  "environment" : "test-env-1",
  "task" : "package",
  "params" : {
    "action" : "install",
    "name" : "httpd"
  },
  "scope" : {
    "nodes" : [ "node1.example.com" ]
  }
}
```

A complete curl request using this body might look like:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/task"
data='{"environment": "test-env-1", "task": "package", "params": {
  "action": "install", "name": "httpd" }, "scope": { "nodes": [
  "node1.example.com" ] }}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

For additional scope examples, refer to the [POST /command/deploy](#) on page 681 request format examples.

Targets

The targets key contains an array of JSON objects, where each object contains the following keys:

Key	Format	Definition
hostnames	Array	Required: An array of hostnames sharing the same target attributes. Each hostname must match an entry in the tasks' node list scope.
user	String	Required: Specify the user on the remote system to use to run the task.
transport	String	Required: Specify ssh or winrm.
password	String	Conditionally required: Specify the password associated with the user key. You must specify either this key or private-key-content.

Key	Format	Definition
private-key-content	String	Conditionally required: Specify the content of the SSH key used to ssh to the remote node to run on. You must specify either this key or password.
port	Integer	Specifies the port, on the remote node, to use to connect.
run-as	String	When using SSH, specify an optional user to use to run commands.
sudo-password	String	If you specify run-as, specify a password to use when changing users.
run-as-command	String	If you specify run-as, specify a command to use to elevate permissions.
connect-timeout	Integer	How long, in seconds, you want Bolt to wait when establishing connections.
tty	Boolean	Whether Bolt uses pseudo tty to meet sudoer restrictions.
tmpdir	String	Specify the directory the task can use to upload and execute temporary files on the target.
extensions	String	A list of file extensions that are accepted for scripts or tasks.

For example, this target array contains two JSON objects:

```
[
  {
    "hostnames": [ "sshnode1.example.com" , "sshnode2.example.com" ] ,
    "private-key-content": "<SSH_KEY>" ,
    "port": 4444 ,
    "user": "<USER_NAME>" ,
    "transport": "ssh"
  } ,
  {
    "hostnames": [ "winrmnode.example.com" ] ,
    "password": "<PASSWORD>" ,
    "port": 4444 ,
    "user": "<USER_NAME>" ,
    "transport": "winrm"
  }
]
```

Response format

If the task starts successfully, the server returns 202 and a JSON object using these keys:

- **id:** An absolute URL that links to the newly created job. You can use it with the [GET /jobs/<job-id>](#) on page 708 endpoint to retrieve information about the status of the job.

- name: A stringified number identifying the newly created job.

For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/81"
    "name" : "81"
  }
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Value	Definition
404	puppetlabs/orchestrator/unknown-environment	The specified environment does not exist.
400	puppetlabs/orchestrator/empty-target	There is a problem with the target entry in the request body, such as the hostnames do not match nodes defined by the scope.
400	puppetlabs/orchestrator/puppetdb-error	If the request specified a query for the scope, the orchestrator is unable to make a query to PuppetDB.
400	puppetlabs/orchestrator/query-error	If the request specified a query for the scope, the query is invalid or the user submitting the request does not have permission to run the query.
403	puppetlabs/orchestrator/not-permitted	This error occurs when a user does not have permission to run the task on the requested nodes.

POST /command/task_target

Create a *task-target*, which is a set of tasks and nodes/node groups you can use to provide specific privilege escalation for users who would otherwise not be able to run certain tasks or run tasks on certain nodes or node groups. When you grant a user permission to use a task-target, the user can run the task(s) in the task-target on the set of nodes defined in the task-target.

Important: After using the POST /command/task_target endpoint to create a task-target, you must use the [POST /roles](#) on page 332 endpoint to create a role controlling the permission to use the task-target. For an overview of the task-target workflow, read about [Puppet Enterprise RBAC API](#), or [how to manage access to tasks](#) on the Puppet blog.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is application/json. The body must be a JSON object using these keys:

Key	Format	Definition
display_name	String	Required: The task-target name. There are no uniqueness requirements.

Key	Format	Definition
tasks	Array of strings	<p>Conditionally required: You must specify either <code>tasks</code> or <code>all_tasks</code>.</p> <p>If you want to include specific tasks in the task-target, use <code>tasks</code> to supply an array of relevant task names. This key can be empty. If <code>tasks</code> is omitted or empty, you must set <code>all_tasks</code> to <code>true</code>. This key is required if <code>all_tasks</code> is omitted.</p> <p>Important: The endpoint does not check if the specified tasks correspond to existing tasks. This means you can create task-targets that include tasks you have not yet created, and that you must manually confirm the task names are spelled correctly.</p>
all_tasks	Boolean	<p>Conditionally required: You must specify either <code>tasks</code> or <code>all_tasks</code>.</p> <p><code>all_tasks</code> indicates whether any tasks can be run on designated node targets. The default is <code>false</code> and expects you to define specific tasks in the <code>tasks</code> key. However:</p> <ul style="list-style-type: none"> • If <code>tasks</code> is omitted or empty, you must set <code>all_tasks</code> to <code>true</code>. • If <code>all_tasks</code> is omitted, you must provide a valid <code>tasks</code> key. • If <code>all_tasks</code> is <code>true</code>, omit <code>tasks</code>. If you specify <code>tasks</code> and set <code>all_tasks</code> to <code>true</code>, the endpoint ignores <code>tasks</code> and takes the <code>all_tasks</code> value.

Key	Format	Definition
nodes	Array of strings	<p>Required: Use nodes, node_groups, and pq1_query to identify nodes users can run tasks against when using this task-target. The endpoint combines these keys to form a total node pool. If you specified tasks, the user can run only those specific tasks against the specified nodes.</p> <p>nodes must be either empty array or an array of certnames identifying specific agent nodes or agentless nodes to associate with this task-target.</p> <p>Important: The endpoint does not check if the nodes certnames correspond to existing nodes. This means you can create task-targets that include individual nodes you have not yet configured, and that you must manually confirm the node certnames are specified correctly.</p>
node_groups	Array of strings	<p>Required: Use nodes, node_groups, and pq1_query to identify nodes users can run tasks against when using this task-target. The endpoint combines these keys to form a total node pool. If you specified tasks, the user can run only those specific tasks against the specified nodes.</p> <p>node_groups must be either an empty array or an array of node group IDs describing node groups associated with this task-target.</p>
pql_query	String	<p>Use nodes, node_groups, and pql_query to identify nodes users can run tasks against when using this task-target. The endpoint combines these keys to form a total node pool. If you specified tasks, the user can run only those specific tasks against the specified nodes.</p> <p>pql_query is an optional string specifying a single PQL query to use to fetch nodes for this task-target. Query results must contain the certnames key to identify the nodes.</p> <p>Important: While pql_query is optional, if you only use pql_query to define the nodes in the task-target, you must supply nodes and node_groups as empty arrays.</p>

For example, this body creates a task target that allows users to run task_1 and task_2 on node1, node2, and all nodes a specific node group.

```
{
  "display_name": "task_target_example_1",
```

```

  "tasks": [ "task_1", "task_2" ],
  "nodes": [ "node1" "node2" ],
  "node_groups": [ "00000000-0000-4000-8000-000000000000" ]
}

```

This body allows users to run any task against node1 and node2.

```

{
  "display_name": "task_target_example_2",
  "all_tasks": "true",
  "nodes": [ "node1" "node2" ],
  "node_groups": []
}

```

A complete curl request for the command/task_target endpoint might look like:

```

type_header='Content-Type: application/json'
auth_header='X-Authentication: $(puppet-access show)'
uri="https://$(puppet config print server):8143/orchestrator/v1/command/
task_target"
data='{"display_name": "task_target_example_2", "all_tasks": "true",
"nodes": [ "node1" "node2" ], "node_groups": []}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"

```

Response format

If the task-target is successfully created, the server returns a 200 response and a JSON object using these keys:

- **id**: An absolute URL that links to the task-target. You can use it with the [GET /scopes/task_targets/<task-target-id>](#) on page 757 endpoint to retrieve information about the task-target.
- **name**: The task-target's unique identifier.

For example:

```

{
  "task_target": {
    "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/1",
    "name": "1"
  }
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. There are several reasons this endpoint might return a `400 puppetlabs.orchestrator/validation-error` response, including:

- `display_name` is missing or empty.
- `tasks` is missing or empty when `all_tasks` is `false` or omitted.
- Task names in `tasks` are not supplied as strings.
- If `all_tasks` is defined, the value is not a Boolean.
- `tasks` is missing or empty and `all_tasks` is not `true`.
- Node names in `nodes` are not supplied as strings.
- Node group IDs in `node_groups` are not supplied as strings.
- The value of `pql_query` is not a string.

Related information

[POST /roles](#) on page 332

Create a role. Authentication is required.

POST /command/plan_run

Use the plan executor to run a plan.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object using these keys:

Key	Format	Definition
plan_name	String	Required: The name of the plan to run. Tip: Use the GET /plans on page 732 endpoint to find plan names.
params	JSON object	The parameters you want the plan to use.
environment	String	The environment to load the plan from. The default is <code>production</code> .
description	String	A description of the job.
userdata	JSON object	Arbitrary key/value data supplied to the job.

For example, this body starts the `canary` plan on two specific nodes:

```
{
  "plan_name" : "canary",
  "description" : "Start the canary plan on node1 and node2",
  "params" : {
    "nodes" : [ "node1.example.com", "node2.example.com" ],
    "command" : "whoami",
    "canary" : 1
  }
}
```

A complete curl request using this body might look like:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/
plan_run"
data='{"plan_name" : "canary", "description" : "Start the canary plan
on node1 and node2", "params" : { "nodes" : [ "node1.example.com",
"node2.example.com" ], "command" : "whoami", "canary" : 1}}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

Response format

If the plan starts successfully, the server returns 202 and a JSON object containing the name of the newly created plan job. For example:

```
{
  "name" : "1234"
}
```

If you need to stop this plan while it is running, use the name ID with [POST /command/stop_plan](#) on page 686.

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The plan_name is not valid. Most likely, it is not a properly-formatted string.
403	puppetlabs/orchestrator/not-permitted	This error occurs when a user does not have permission to run the plan, cannot run the plan on the specified nodes, or otherwise lacks permission to complete the request.

POST /command/environment_plan_run

Use parameters to run a plan on specific nodes in a specific environment.

Request format

This endpoint is similar to the [POST /command/plan_run](#) on page 695 endpoint. You must define nodes to run the plan on by supplying parameters in your request.

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is application/json. The body must be a JSON object using these keys:

Key	Format	Definition
plan_name	String	Required: The name of the plan to run. Tip: Use the GET /plans on page 732 endpoint to find plan names.
params	JSON object containing objects	Required: The parameters you want the plan to use. Use the type key to identify whether a parameter is a PuppetDB query or a node group.
environment	String	The environment to load the plan from. The default is production.
description	String	A description of the job.
userdata	JSON object	Arbitrary key/value data supplied to the job.

For example, this body runs the `example_plan` on nodes specified in the `targets` and `more_targets` parameters:

```
{
  "plan_name" : "example_plan",
  "description" : "Output 'message' on the targets contained in 'targets' and 'more targets'",
  "params": {
    "message": {
      "value": "hello"
    },
    "example_object_param": {
      "value": {
        "value": "xyz"
      }
    },
    "targets": {
      "type": "query",
      "value": "nodes[certname] { }"
    },
    "more_targets": {
      "type": "node_group",
      "value": "<UUID>"
    }
  }
}
```

In this example, the parameters follow the format `{<PARAM_NAME>: { "value": "<PARAM_VALUE>" }}`, such as `{ "message": { "value": "hello" }}`. The orchestrator passes `<PARAM_VALUE>` to the plan as the parameter's value. For object parameters, such as `example_object_param`, the object `{ "value": "xyz" }` is passed to the plan as the value for `example_object_param`.

You can use the optional `type` key to give the orchestrator additional information about the parameter. This tells the orchestrator how to interpret the parameter's value. It has no relationship to the parameter's type in the plan metadata. The `type` key accepts these values:

- `query`: The parameter's value in the request is interpreted as a PuppetDB query. Orchestrator executes the query and passes the resulting list of nodes to the plan as the parameter's value. For example, if the `targets` parameter is set to `nodes[certname] { }` and resolves to `["node1", "node2"]`, orchestrator passes `["node1", "node2"]` into the plan as the parameter's value.
- `node_group`: The parameter's value in the request is interpreted as a node group UUID. Orchestrator fetches all the nodes in the node group and passes them to the plan as the parameter's value. For example, if the `more_targets` parameter is set to a specific UUID and resolves to `["node_group1", "node_group2"]`, orchestrator passes that array into the plan as the parameter's value.

Refer to [POST /command/plan_run](#) on page 695 for curl command examples you can modify to use with this endpoint.

Response format

If the plan starts successfully, the server returns 202 and a JSON object containing the name of the newly created plan job. For example:

```
{
  "name" : "1234"
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the `kind` key:

Response code	Key	Description
400	puppetlabs.orchestrator/validation-error	Any of various possible validation errors, such as the plan_name not being a properly-formatted string or an error in the params.
403	puppetlabs.orchestrator/not-permitted	This error occurs when a user does not have permission to run the plan, cannot run the plan on the specified nodes, or otherwise lacks permission to complete the request.

POST /command/schedule_deploy (deprecated)

Prior to deprecation, this endpoint scheduled a Puppet run on a set of nodes.

Important: This endpoint is deprecated. Instead, use [POST /scheduled_jobs/environment_jobs](#) on page 724.

Request format

Prior to deprecation, requests to this endpoint:

- Specified the content type as application/json
- Contained a JSON object body
- Included these required keys: environment, scope, scheduled_time
- Could include several optional keys.

For example, this request scheduled a deployment targeting nodes in the All Nodes node group in the production environment:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/schedule_deploy"
data='{"environment": "production", "scope" : { "node_group" :
"00000000-0000-4000-8000-000000000000" }, "scheduled_time":
"2027-05-05T19:50:08Z"}'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

Response format

Prior to deprecation, if the deployment was successfully scheduled, the server returned 202 and a JSON object containing the job's id and name. For example:

```
{
  "job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/81"
    "name" : "81"
  }
}
```

POST /command/schedule_task (deprecated)

Prior to deprecation, this endpoint scheduled a task to run at a future date and time.

Important: This endpoint is deprecated. Instead, use [POST /scheduled_jobs/environment_jobs](#) on page 724.

Request format

Prior to deprecation, requests to this endpoint:

- Specified the content type as `application/json`
- Contained a JSON object body
- Included these required keys: `environment`, `params`, `scope`, `scheduled_time`, and `task`
- Could include several optional keys.

For example, this request scheduled the package task to run on nodes in the All Nodes node group in the production environment and passed no parameters to the task:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/schedule_task"
data='{"environment" : "production", "task" : "package", "params" : {}, "scope" : { "node_group" : "00000000-0000-4000-8000-000000000000" }, "scheduled_time": "2027-05-05T19:50:08Z"}'

curl --insecure --header "$type_header" --header "$auth_header" --request POST "$uri" --data "$data"
```

Response format

Prior to deprecation, if the task was successfully scheduled, the server returned 202 and a JSON object containing the job's id and name. For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/81"
    "name" : "81"
  }
}
```

POST /command/schedule_plan (deprecated)

Schedule a plan to run at a later time.

Important: This endpoint is deprecated. Instead, use [POST /scheduled_jobs/environment_jobs](#) on page 724.

Request format

Prior to deprecation, requests to this endpoint:

- Specified the content type as `application/json`
- Contained a JSON object body
- Included these required keys: `plan_name` and `scheduled_time`
- Could include some optional keys.

For example, this request scheduled the canary plan to run on two specific nodes:

```
type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/command/schedule_plan"
data='{"plan_name" : "canary", "params" : { "nodes" : [ "node1.example.com", "node2.example.com" ]}, "scheduled_time" : "2027-05-05T19:50:08Z"}'
```

```
curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"
```

Response format

Prior to deprecation, if the plan was successfully scheduled, the server returned 202 and a JSON object containing the job's id and name. For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/81",
    "name" : "81"
  }
}
```

Inventory endpoints

Use the inventory endpoints to check whether the orchestrator can reach a node.

These endpoints are based on nodes being connected to the Puppet Communications Protocol (PCP) broker, which is part of the [Puppet orchestrator architecture](#).

GET /inventory

Retrieve a list of all nodes connected to the Puppet Communications Protocol (PCP) broker.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the request is a basic call, such as:

```
GET https://orchestrator.example.com:8143/orchestrator/v1/inventory
```

The GET /inventory endpoint does not support any parameters; however, as with other orchestrator API endpoints, you must provide authentication.

Response format

A successful response is a JSON object containing an array of nodes. The response uses the following keys to provide information about each node's PCP broker connection:

Key	Definition
name	The node's name.
connected	The status of the connection between the node and the PCP broker, either <code>true</code> (connected) or <code>false</code> (disconnected).
broker	The PCP broker the node is connected to. If <code>connected</code> is <code>false</code> , this key is empty or omitted.
timestamp	The time when the node connected to the PCP broker. If <code>connected</code> is <code>false</code> , this key is empty or omitted.

For example, this response provides details about three nodes, one of which is currently disconnected:

```
{
  "items" : [
    {
      "name" : "node1.example.com",
      "connected" : true,
```

```

    "broker" : "pcp://broker1.example.com/server",
    "timestamp": "2016-010-22T13:36:41.449Z"
},
{
    "name" : "node2.example.com",
    "connected" : true,
    "broker" : "pcp://broker2.example.com/server",
    "timestamp" : "2016-010-22T13:39:16.377Z"
},
{
    "name" : "node3.example.com",
    "connected" : false
}
]
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 500 response if the PCP broker can't be reached.

GET /inventory/<node>

Retrieve information about a single node's connection to the Puppet Communications Protocol (PCP) broker.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include a specific node name, such as:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/inventory/
<NODE_NAME>"
```

If you do not know the node's name, you can use the [GET /inventory](#) on page 700 endpoint to query all nodes. If you want to query multiple specific nodes at once, use [POST /inventory](#) on page 702.

The GET /inventory/<node> endpoint does not support any additional parameters; however, as with other orchestrator API endpoints, you must provide authentication.

Response format

A successful response is a JSON object that uses these keys to provide information about the specified node's PCP broker connection:

Key	Definition
name	The node's name.
connected	The status of the connection between the node and the PCP broker, either <code>true</code> (connected) or <code>false</code> (disconnected).
broker	The PCP broker the node is connected to. If <code>connected</code> is <code>false</code> , this key is empty or omitted.
timestamp	The time when the node connected to the PCP broker. If <code>connected</code> is <code>false</code> , this key is empty or omitted.

For example:

```
{
    "name" : "node1.example.com",
```

```

"connected" : true,
"broker" : "pcp://broker.example.com/server",
"timestamp" : "2017-03-29T21:48:09.633Z"
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 500 response if the PCP broker can't be reached.

POST /inventory

Returns information about multiple nodes' connections to the Puppet Communications Protocol (PCP) broker.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is application/json. The body must be a JSON object specifying an array node names, such as:

```
{
  "nodes" : [
    "node1.example.com",
    "node2.example.com",
    "node3.example.com"
  ]
}
```

A complete curl request using this body might look like:

```

$type_header='Content-Type: application/json'
$auth_header="X-Authentication: $(puppet-access show)"
$uri="https://$(puppet config print server):8143/orchestrator/v1/inventory"
$data='{"nodes" : [ "node1.example.com", "node2.example.com",
  "node3.example.com"] }'

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"

```

Response format

A successful response is a JSON object that uses these keys to provide information about each node's PCP broker connection:

Key	Definition
name	The node's name.
connected	The status of the connection between the node and the PCP broker, either true (connected) or false (disconnected).
broker	The PCP broker the node is connected to. If connected is false, this key is empty or omitted.
timestamp	The time when the node connected to the PCP broker. If connected is false, this key is empty or omitted.

For example, this is a response to a query about three specific nodes:

```
{
  "items" : [
    {
      "
```

```

        "name" : "node1.example.com",
        "connected" : true,
        "broker" : "pcp://broker.example.com/server",
        "timestamp" : "2017-07-14T15:57:33.640Z"
    },
    {
        "name" : "node2.example.com",
        "connected" : false
    },
    {
        "name" : "node3.example.com",
        "connected" : true,
        "broker" : "pcp://broker.example.com/server",
        "timestamp" : "2017-07-14T15:41:19.242Z"
    }
]
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 500 response if the PCP broker can't be reached.

Jobs endpoints

Use the jobs endpoints to examine jobs and their details.

You can:

- [GET /jobs](#) on page 703: Retrieve details about all known jobs.
- [GET /jobs/<job-id>](#) on page 708: Retrieve details about a specific job.
- [GET /jobs/<job-id>/nodes](#) on page 710: Retrieve information about nodes associated with a specific job.
- [GET /jobs/<job-id>/report](#) on page 714: Retrieve a summary of a specific job.
- [GET /jobs/<job-id>/events](#) on page 715: Retrieve a list of events that occurred during a specific job.

For details about plan jobs, use the [Plan jobs endpoints](#) on page 736.

To stop an in-progress job, use [POST /command/stop](#) on page 685.

GET /jobs

Retrieve details about all jobs that the orchestrator knows about.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can append parameters to the end of the URI path, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs?
limit=20&offset=20
```

These parameters are available:

Parameter	Definition
limit	Set the maximum number of jobs to include in the response. The point at which the limit count starts is determined by offset, and the job record sort order is determined by order_by and order.

Parameter	Definition
offset	Specify a zero-indexed integer at which to start returning results. For example, if you set this to 12, the response returns jobs starting with the 13th record. The default is 0.
order_by	Specify one of the following categories to use to sort the results: owner, timestamp, environment, name, or state. Sorting by owner uses the login subfield of owner records.
order	Indicate whether results are returned in ascending (asc) or descending (desc) order. The default is asc.
type	Specify a job type to query, either deploy, task, or plan_task.
task	Specify a task name to match. Partial matches are supported. If you specified type=deploy, you can't use task.
min_finish_timestamp	Returns only the jobs that finished at or after the supplied UTC timestamp.
max_finish_timestamp	Returns only the jobs that finished at or before the supplied UTC timestamp.

Response format

The response is a JSON object containing an array, called `items`, and an object, called `pagination`.

The `items` array contains a JSON object for each job. Each object uses these keys to provide job details:

Key	Definition
id	An absolute URL that links to the newly created job.
name	A stringified number identifying the newly created job. You can use this with other endpoints, such as GET /jobs/<job-id> on page 708 (retrieve information about the status of the job) and POST /command/stop on page 685.
state	The job's current state: new, ready, running, stopping, stopped, finished, or failed
<p>Tip: If you want to know when a job entered and exited each state, use the GET /jobs/<job-id> on page 708 endpoint.</p>	
command	The command that created the job.
type	The job type: deploy, task, plan_task, plan_script, plan_upload, plan_command, plan_wait, plan_apply, plan_apply_prep

Key	Definition
options	Options used to create the job (based on the command), a description of the job, and the environment the job operated in.
	Previously, the description and environment key were separate from options. However, these are deprecated. Refer to the keys in options instead.
owner	The subject ID, login, and other details of the user that requested the job.
timestamp	The time when the job's state last changed.
started_timestamp	The time the job was created and started.
finished_timestamp	The time the job finished.
duration	If the job is finished, this is the number of seconds the job took to run. If the job is still running, this is the number of seconds the job has been running.
node_count	The number of nodes the job ran on.
node_states	A JSON map containing the number of nodes involved with the job categorized by current node state. States with no nodes are omitted. If there were no nodes associated with the job, this value is null.
nodes	A link to get more information about the nodes participating in a given job. You can use this with the GET /jobs/<job-id>/nodes on page 710 endpoint.
report	A link to the report for a given job. You can use this with the GET /jobs/<job-id>/report on page 714 endpoint.
events	A link to the events for a given job. You can use this with the GET /jobs/<job-id>/events on page 715 endpoint.
userdata	An object of arbitrary key/value data supplied to the job.

The pagination object uses these keys:

- `total`: The total number of job records in the collection, regardless of `limit` and `offset`.
- `limit`, `offset`, `order_by`, `order`, and `type`: Reflects values supplied in the request. If you specified a value, these key shows the value you specified. If you did not specify a value, the key shows the default value.

Here is an example response describing two jobs and pagination information:

```
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/jobs/1234",
      "name": "1234",
      "state": "finished",
      "command": "deploy",
      "type": "deploy",
      "node_count": 5,
    }
  ],
  "total": 2
}
```

```

"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

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 400 `puppetlabs.orchestrator/validation-error` response if there is a problem with a supplied parameter, such as the `limit` parameter not being formatted as an integer.

GET /jobs/<job-id>

Retrieve details of a specific job, including the start and end times for each job state.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific task or deployment. For example, this request queries a job with ID 375:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/375
```

Job IDs are returned in responses from some [Command endpoints](#) on page 681 and [GET /jobs](#) on page 703.

A complete request might look like:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/jobs/81"
curl --insecure --header "$auth_header" "$uri"
```

Response format

The response is a JSON object that uses these keys to provide job details:

Key	Definition
<code>id</code>	The URI path from the request.
<code>name</code>	A stringified number identifying the job.
<code>state</code>	The job's current state: new, ready, running, stopping, stopped, finished, or failed
	Tip: The <code>status</code> key shows when the job entered and exited each state.
<code>command</code>	The command that created the job.
<code>type</code>	The job type: deploy, task, plan_task, plan_script, plan_upload, plan_command, plan_wait, plan_apply, plan_apply_prep
<code>options</code>	Options used to create the job (specific options depend on the command), a description of the job, and the environment the job operated in.
	Previously, the <code>description</code> and <code>environment</code> key were separate from <code>options</code> . However, these are deprecated. Refer to the keys in <code>options</code> instead.

Key	Definition
owner	The subject ID and login of the user that requested the job.
timestamp	The time when the job's state last changed.
started_timestamp	The time the job was created and started.
finished_timestamp	The time the job finished.
duration	If the job is finished, this is the number of seconds the job took to run. If the job is still running, this is the number of seconds the job has been running.
node_count	The number of nodes the job ran on.
nodes	A link to get more information about the nodes participating in a given job. You can use this with the GET /jobs/<job-id>/nodes on page 710 endpoint.
report	A link to the report for a given job. You can use this with the GET /jobs/<job-id>/report on page 714 endpoint.
events	A link to the events for a given job. You can use this with the GET /jobs/<job-id>/events on page 715 endpoint.
status	The start and end times for each state the job was in.
userdata	An object of arbitrary key/value data supplied to the job.

Here is an example response:

```
{
  "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" : {
      "nodes" : [ "node1.example.com", "node2.example.com" ] ,
      "enforce_environment" : true,
      "environment" : "production",
      "evaltrace" : false,
      "target" : null
    },
    "node_count" : 2,
    "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-4cccd-9f4f-e93db6aa38ec",
      "role_ids" : [ 1 ],
      "display_name" : "Admin",
      "password" : "P@ssw0rd"
    }
  }
}
```

```

    "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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs.orchestrator/validation-error	The job ID in the request is not an integer.
404	puppetlabs.orchestrator/unknown-job	No job exists that matches the specified job ID.

GET /jobs/<job-id>/nodes

Retrieve information about nodes associated with a specific job.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific task or deployment. Job IDs are returned in responses from some [Command endpoints](#) on page 681 and [GET /jobs](#) on page 703.

You can also append these optional parameters:

Key	Definition
limit	Set the maximum number of nodes to include in the response. The point at which the limit count starts is determined by offset, and the node record sort order is determined by order_by and order.
offset	Specify a zero-indexed integer at which to start returning results. For example, if you set this to 12, the response returns nodes starting with the 13th record. The default is 0.
order_by	Specify one of the following categories to use to sort the results: name, duration, state, start_timestamp, or finish_timestamp.
order	Indicate whether results are returned in ascending (asc) or descending (desc) order. The default is asc.
state	Specify a specific node state to query: new, ready, running, stopping, stopped, finished, or failed.

For example, this URI queries up to 20 nodes in failed status:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/375/nodes?
limit=20&state=failed
```

Response format

The response is a JSON object containing an object, called `next-events`, and an array, called `items`.

The `next-events` object contains these keys:

- `id`: A URI path you can use with the [GET /jobs/<job-id>/events](#) on page 715 endpoint to get information about events associated with the job specified in the request.
- `event`: An event ID.

The `items` array contains a JSON object for each node associated with the job. Each object uses these keys to provide node details:

Key	Definition
<code>timestamp</code>	The time of the most recent activity on the node. This is deprecated; use <code>start_timestamp</code> and <code>finish_timestamp</code> instead.
<code>start_timestamp</code>	The time the node starting running. If the node hasn't started running, or if it was skipped, the value is <code>nil</code> .
<code>finish_timestamp</code>	The time the node finished running. If the node hasn't finished running, or if it was skipped, the value is <code>nil</code> .

Key	Definition
duration	<ul style="list-style-type: none"> If the node has finished running, this value is the duration, in seconds, of the Puppet run. If the node is currently running, this value is the time, in seconds, that has elapsed since the node started running. If the node hasn't started or was skipped, the value is nil.
state	The node's current state.
transaction_uuid	The ID of the nodes last report. This field is deprecated.
name	The node's hostname.
details	<p>A JSON object containing a message and other information about the node's last event and the node's current state. It can be empty, and it might duplicate information from the results for historical reasons.</p> <p>For example:</p> <ul style="list-style-type: none"> If the node's state is finished or failed, the details include a report-url. If the node's state is errored or skipped, check the message for information about the problem. If the node's state is running, the details include the run-time in seconds.
result	A JSON object describing the outcome and event information from the last node run. Exact contents depends on the job type (task job or a Puppet run) and whether the run succeeded, failed, or encountered an error. The node run must have ended to report results. For a task job, the results reflect the outcome of executing the task. For a Puppet run, the results reflect metrics from the Puppet run.

For example, this response describes one node:

```
{
  "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" : "finished",
  }
]
```

```

"transaction_uuid" : <UUID>,
"name" : "node1.example.com",
"details" : {
    "message": "Message from latest event"
},
"result" : {
    "output_1": "success",
    "output_2": [1, 1, 2, 3]
}
}
}

```

Here are some examples of various `results` objects:

Errors when running tasks

```

"result" : {
    "msg" : "Running tasks is not supported for agents older than version
5.1.0",
    "kind" : "puppetlabs.orchestrator/execution-failure",
    "details" : {
        "node" : "copper-6"
    }
}

"result" : {
    "error" : "Invalid task name 'package::status'"
}

```

Raw, standard task run output

```

"result" : {
    "output" : "test\n"
}

```

Structured task run output

```

"result" : {
    "status" : "up to date",
    "version" : "5.0.0.201.g879fc5a-1.el7"
}

```

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"
}
```

puppet-apply ran as part of a plan

```

"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
  }
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The job ID in the request is not an integer.
404	puppetlabs/orchestrator/unknown-job	No job exists that matches the specified job ID.

GET /jobs/<job-id>/report

Returns a report that summarizes a specific job.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific task or deployment. For example, this request queries a job with ID 375:

```
https://orchestrator.example.com:8143/orchestrator/v1/jobs/375/report
```

Job IDs are returned in responses from some [Command endpoints](#) on page 681 and [GET /jobs](#) on page 703.

Response format

The response is a JSON object containing an array called items. The array contains one or more JSON objects summarizing the job's status on each node it ran (or is currently running) on. These keys are used:

Key	Definition
node	The hostname of a node that the job ran on.
state	The job's current state on a specific node: new, ready, running, stopping, stopped, finished, or failed
timestamp	The time when the job's state last changed.

Key	Definition
events	IDs of events associated with the particular job and node.

For example, this response contains one summary report:

```
{
  "items" : [ {
    "node" : "node1.example.com",
    "state" : "running",
    "timestamp" : "2015-07-13T20:37:01Z",
    "events" : [ ]
  }, {
    ...
  } ]
}
```

If you want more details about the job, use the [GET /jobs/<job-id>](#) on page 708 endpoint.

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The job ID in the request is not an integer.
404	puppetlabs/orchestrator/unknown-job	No job exists that matches the specified job ID.

GET /jobs/<job-id>/events

Retrieve a list of events that occurred during a specific job.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific task or deployment. Job IDs are returned in responses from some [Command endpoints](#) on page 681 and [GET /jobs](#) on page 703.

You can use the optional start parameter to start the list of events from a specific event ID number.

For example, this request queries events associated with the 352 job, starting with event number 1272:

```
GET https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?
start=1272
```

Response format

A successful response is a JSON object that uses these keys to detail the events in the job:

Key	Definition
next-events	Contains the id subkey, which has the URI supplied in the request as its value.
id	An event's ID.
items	An array of JSON objects where each object is an event.

Key	Definition
type	Each event has one event type, determined by the event's status or circumstances that cause it to occur: <ul style="list-style-type: none"> node_errorred: There was an error running Puppet on a node. node_failed: The Puppet run failed. node_finished: Puppet ran successfully. node_running: Puppet has started running on a node. node_skipped: A Puppet run was skipped on a node (for example, when a dependency fails). job_aborted: A job ended without completing. job_stopping: The job received a stop request, but it is still running (in the process of stopping). job_finished: The job is no longer running. The details describe the final outcome. In the absence of errors, job_finished is always the last event for any job.
timestamp	The time when the event was created.
details	A JSON object containing information about the event.
message	A message about the event.

Here is an example response body:

```
{
  "next-events" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/jobs/352/events?start=1272"
  },
  "items" : [ {
    "id" : "1272",
    "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 ... "
  }]
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the `kind` key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The job ID or the <code>start</code> parameter in the request are not supplied as integers.
404	puppetlabs/orchestrator/unknown-job	No job exists that matches the specified job ID.

Scheduled jobs endpoints

Use the `scheduled_jobs` endpoints to query, edit, and delete scheduled orchestrator jobs.

You can:

- [GET /scheduled_jobs/environment_jobs](#) on page 717: Retrieve information about all scheduled environment jobs.
- [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721: Retrieve information about a single scheduled environment job.
- [POST /scheduled_jobs/environment_jobs](#) on page 724: Create an environment job to run in the future.
- [PUT /scheduled_jobs/environment_jobs/<job-id>](#) on page 730: Edit or delete an existing scheduled environment job.

Tip: An *environment job* is a deployment, task, or plan that runs in a specific environment, such as your production environment.

GET /scheduled_jobs/environment_jobs

Retrieve information about scheduled environment jobs, which are deployments, tasks, or plans that run in a specific environment. Use parameters to narrow the response scope.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can append optional parameters to the end of the URI path, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/
environment_jobs?limit=50&type=task
```

These parameters are available:

Parameter	Definition
<code>limit</code>	Set the maximum number of scheduled jobs to include in the response. The point at which the <code>limit</code> count starts is determined by <code>offset</code> , and the job record sort order is determined by <code>order_by</code> and <code>order</code> .
<code>offset</code>	Specify a zero-indexed integer at which to start returning results. For example, if you set this to 12, the response returns scheduled jobs starting with the 13th record. The default is 0.
<code>order_by</code>	Specify one of the following categories to use to sort the results: <code>next_run_time</code> , <code>environment</code> , <code>owner</code> , <code>name</code> , or <code>type</code> . The default is <code>next_run_time</code> .

Parameter	Definition
order	Indicate whether results are returned in ascending (<code>asc</code>) or descending (<code>desc</code>) order. The default is <code>asc</code> .
type	Specify a job type to query, either <code>deploy</code> , <code>task</code> , or <code>plan</code> .

Response format

The response is a JSON object containing an `items` array and a `pagination` object.

The `items` array contains a JSON object for each scheduled job. Each object can use these keys to provide job details:

Key	Definition
<code>id</code>	The job's absolute URL, which includes the job's numerical ID.
<code>name</code>	The job's numerical ID as a stringified number.
<code>enabled</code>	A Boolean indicating whether the job is enabled.
<code>environment</code>	The environment that the job operates in.
<code>owner</code>	The subject ID, email, and other details of the user that requested the job.
<code>description</code>	A user-provided description of the job. This can be empty.
<code>type</code>	The job's type, either <code>plan</code> , <code>task</code> , or <code>deploy</code> .
<code>input</code>	An object describing options supplied to the job, including: <ul style="list-style-type: none"> <code>name</code>: String-formatted project name of the task or plan. <code>parameters</code>: An object containing key-value pairs of non-sensitive parameters and values supplied to the job. <code>sensitive_parameters</code>: An array of names of sensitive parameters supplied to the job.
<code>userdata</code>	An object containing arbitrary key-value pairs supplied by the initiating user, if any were supplied.
<code>schedule</code>	An object describing the job's schedule, including: <ul style="list-style-type: none"> <code>start_time</code>: An ISO-8601 timestamp indicating the first time the job ran (or will run). <code>interval</code>: An object representing the frequency at which the job runs, such as every 1300 seconds.

Key	Definition
next_run	An object containing an ISO-8601 timestamp indicating the scheduled job's next run time.
last_run	If the job has not yet run, this is null. If the job has run, this is an object containing an ISO-8601 timestamp of the scheduled job's most recent run time and a job object.
	The job object can contains the submitted job's id or name if the job succeeded. If the job failed, it contains a submission_errorobject describing why the job failed.

The pagination object includes these keys:

- total: The total number of job records in the collection, regardless of limit and offset.
- limit, offset, order_by, order, and type: Reflects values supplied in the request. If you specified a value, these key shows the value you specified. If you did not specify a value, the key shows the default value, if there is one.

This sample response describes two scheduled jobs and the pagination information:

```
{
  "items": [
    {
      "id": "https://host.example.com:8143/orchestrator/v1/scheduled_jobs/environment_jobs/2",
      "name": "2",
      "environment": "plan_testing_env",
      "owner": {
        "email": "fred@example.com",
        "login": "fred",
        "display_name": "Fred",
        "id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
        "is_revoked": false,
        "last_login": "2020-05-08T15:57:28.444Z",
        "is_remote": true,
        "is_group": false,
        "is_superuser": false,
        "role_ids": [
          1
        ],
        "inherited_role_ids": [
          2
        ],
        "group_ids": [
          "9a588fd8-3daa-4fc2-a396-bf88945def1e"
        ]
      },
      "description": "Fred's scheduled environment plan",
      "type": "plan",
      "next_run": {
        "time": "2021-12-12T19:50:08Z"
      },
      "last_run": {
        "time": "2021-11-12T19:50:08Z",
        "job": {
          "id": "https://host.example.com:8143/orchestrator/v1/plan_jobs/42",
        }
      }
    }
  ],
  "total": 2,
  "limit": 2,
  "offset": 0,
  "order_by": "last_run",
  "order": "desc"
}
```

```

        "name": 42
    }
},
"schedule": {
    "start_time": "2018-10-05T19:50:08Z",
    "interval": {
        "value": 3600,
        "units": "seconds"
    }
},
"input": {
    "name": "plan_testing_env::example_plan",
    "parameters": {
        "param_1": "foo"
    },
    "sensitive_parameters": [ "password" ]
},
"userdata": {
    "ticket": "TICKET-123"
}
},
{
    "id": "https://host.example.com:8143/orchestrator/v1/scheduled_jobs/environment_jobs/1",
    "name": "1",
    "environment": "plan_testing_env",
    "owner": {
        "email": "user@example.com",
        "login": "user",
        "display_name": "User",
        "id": "06990bb9-df3a-4150-964f-88b9cf0f8eec",
        "last_login": "2019-07-08T15:57:28.444Z",
        "is_revoked": false,
        "is_remote": true,
        "is_group": false,
        "is_superuser": false,
        "role_ids": [
            1
        ],
        "inherited_role_ids": [
            2
        ],
        "group_ids": [
            "9a588fd8-3daa-4fc2-a396-bf88945def1e"
        ]
    },
    "description": "",
    "type": "plan",
    "schedule": {
        "start_time": "2018-10-05T19:50:08Z",
        "interval": null
    },
    "input": {
        "name": "plan_testing_env::example_plan",
        "parameters": {
            "param_1": "foo"
        },
        "sensitive_parameters": [ ]
    },
    "userdata": {
        "approval_reference": "442"
    },
    "start_time": "2020-10-05T19:50:08Z",
    "next_run": {

```

```
        "time": "2021-12-12T19:50:08Z"
    },
    "last_run": {
        "time": "2021-11-12T19:50:08Z",
        "job": {
            "submission_error": {
                "kind": "puppetlabs.orchestrator/unknown-environment",
                "msg": "Unknown environment doesnotexist",
                "details": {
                    "environment": "doesnotexist"
                }
            },
            "name": 33
        }
    }
},
"pagination": {
    "limit": 50,
    "offset": 0,
    "order": "asc",
    "order_by": "next_run_time",
    "total": 2,
}
}
```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758.

The endpoint returns a 400 `puppetlabs/orchestrator/query-error` response if you don't have permission to run queries or there is a problem with a supplied parameter, such as the `limit` or `offset` parameters not being formatted as integers.

GET /scheduled_jobs/environment_jobs/<job-id>

Retrieve information about a specific scheduled environment job, which is deployment, task, or plan that runs in a specific environment.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include the ID of the scheduled job you want to query. For example, this request queries a scheduled job with ID 81:

https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/environment_jobs/81

You can use the [GET /scheduled_jobs/environment_jobs](#) on page 717 endpoint to get scheduled job IDs.

Response format

The response is a JSON object that uses these keys to provide the scheduled job's details:

Key	Definition
id	The job's absolute URL.
name	A stringified number identifying the job.
enabled	A Boolean indicating whether the job is enabled.
environment	The environment that the job operates in.

Key	Definition
owner	The subject ID, email, and other details of the user that requested the job.
description	A user-provided description of the job. This can be empty.
type	The job's type, such as <code>plan</code> , <code>task</code> , or <code>deploy</code> .
input	An object describing options supplied to the job, including: <ul style="list-style-type: none"> <code>name</code>: String-formatted project name of the task or plan. <code>parameters</code>: An object containing key-value pairs of non-sensitive parameters and values supplied to the job. <code>sensitive_parameters</code>: An array of names of sensitive parameters supplied to the job.
userdata	An object containing arbitrary key-value pairs supplied by the initiating user, if any were supplied.
schedule	An object describing the job's schedule, including: <ul style="list-style-type: none"> <code>start_time</code>: An ISO-8601 timestamp indicating the first time the job ran (or will run). <code>interval</code>: An object representing the frequency at which the job runs, such as every 1300 seconds.
next_run	An object containing an ISO-8601 timestamp indicating the scheduled job's next run time.
last_run	If the job has not yet run, this is <code>null</code> . If the job has run, this is an object containing an ISO-8601 timestamp of the scheduled job's most recent run time and a <code>job</code> object.
	The <code>job</code> object can contain the submitted job's <code>id</code> or <code>name</code> if the job succeeded. If the job failed, it contains a <code>submission_error</code> object describing why the job failed.

For example:

```
{
  "name": "https://host.example.com:8143/orchestrator/v1/scheduled_jobs/environment_jobs/2",
  "id": "2",
  "environment": "production",
  "owner": {
    "email": "fred@example.com",
    "name": "Fred"
  }
}
```

```

"login": "fred",
"display_name": "Fred",
"id": "784beba4-8cc8-414f-aab0-e9a29c9b65c2",
"is_revoked": false,
"last_login": "2020-05-08T15:57:28.444Z",
"is_remote": true,
"is_group": false,
"is_superuser": false,
"role_ids": [
    1
],
"inherited_role_ids": [
    2
],
"group_ids": [
    "9a588fd8-3daa-4fc2-a396-bf88945def1e"
]
},
"description": "Fred's scheduled environment plan",
"type": "plan",
"next_run": {
    "time": "2021-12-12T19:50:08Z"
},
"last_run": {
    "time": "2021-11-12T19:50:08Z",
    "job": {
        "id": "https://host.example.com:8143/orchestrator/v1/plan_jobs/42",
        "name": 42
    }
},
"schedule": {
    "start_time": "2018-10-05T19:50:08Z",
    "interval": {
        "value": 3600,
        "units": "seconds"
    }
},
"input": {
    "name": "example_module::example_plan",
    "parameters": {
        "param_1": "foo"
    },
    "sensitive_parameters": ["password"]
},
"userdata": {
    "ticket": "TICKET-123"
}
}
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758.

The endpoint returns a `400 puppetlabs/orchestrator/query-error` response if you don't have permission to run queries or the job ID is invalid.

POST /scheduled_jobs/environment_jobs

Create an environment job to run in the future. An environment job is a deployment, task, or plan that runs in a specific environment, such as a Puppet run on nodes in your production environment.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is `application/json`. The body must be a JSON object using these required keys:

Key	Definition
<code>type</code>	Enumerated value indicating the type of action you want to schedule, either <code>plan</code> , <code>task</code> , or <code>deploy</code> .
<code>input</code>	An object describing job parameters, scope, or targets. The contents depends on the <code>type</code> , as described in the <code>input</code> object section, below.
<code>environment</code>	A string specifying the name of the relevant environment. For <code>task</code> and <code>plan</code> jobs, this is the environment from which to load the task or plan. For <code>deploy</code> jobs, this can be an empty string or the name of the environment to deploy.
<code>schedule</code>	An object that uses the <code>start_time</code> and <code>interval</code> keys to describe the job's schedule. The <code>start_time</code> key accepts an ISO-8601 timestamp indicating the first time that you want the job to run. The <code>interval</code> key accepts either an object or <code>null</code> . To only run the job once, use <code>null</code> . To schedule a recurring job, supply an object containing <code>value</code> and <code>units</code> . The <code>units</code> key is an enum that must be set to <code>seconds</code> . The <code>value</code> key is an integer representing the number of <code>units</code> to wait between job runs. For example: <pre>"interval": { "units": "seconds", "value": 86400 }</pre>
<code>description</code>	A string describing the job. You can supply an empty string.

Key	Definition
userdata	An object containing arbitrary key-value pairs supplied to the job, such as a support ticket number. You can supply an empty object.

For example, this request schedules the facts plan to run once on the node called `node1.example.com` in an environment called `my_environment`:

```

type_header='Content-Type: application/json'
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/
scheduled_jobs/environment_jobs"
data='
{
  "description": "run facts plan once on node1 in my_environment",
  "input": {
    "name": "sensitive_params::scheduled_jobs_storage",
    "parameters": {
      "primary": {
        "value": "example.delivery.puppetlabs.net"
      },
      "param_one": {
        "value": "first_value"
      }
    },
    "environment": "my_environment",
    "schedule": {
      "start_time": "2022-01-28T09:35:56-08:00",
      "interval": null
    },
    "userdata": {
      "snow_ticket": "INC0011211"
    },
    "type": "plan"
  }
}

curl --insecure --header "$type_header" --header "$auth_header" --request
POST "$uri" --data "$data"

```

The input object

The contents of the `input` object depends on the job's type (either `plan`, `deploy`, or `task`). The keys you can use in the `input` object for each job type are described below. Not all keys are required, but the `input` object itself is a required key.

The `input` object for `plan` jobs can contain:

- `name`: String-formatted name of the plan to run.
- `parameters`: An object containing a series of key-value pairs representing parameter inputs for the plan to use. Parameters either specify standard parameter inputs or node targets.

Standard parameter inputs are represented as a parameter name key with the `value` in an object. The `value` key accepts any data type as long as it is valid JSON. For example:

```

"parameters": {
  "param_one": {
    "value": "some_value"
}

```

```
}
```

The plan's node targets can be declared in the `targets` object. You can declare specific nodes in the `value` key, much like a standard parameter input, or you can declare a variable group of nodes by supply a PuppetDB query. To do this, you must supply a `targets` object that contain your query statement in the `value` key and `"type": "query"`. For example:

```
"parameters": {
    "targets": {
        "type": "query",
        "value": "nodes[certname] { }"
    }
}
```

Supplying `"type": "query"` instructs the orchestrator to interpret the `value` as a PuppetDB query and pass the resulting node lists as the parameter's value.

Here is a complete example of an `input` object for a `plan` job:

```
"input": {
    "name": "sensitive_params::scheduled_jobs_storage",
    "parameters": {
        "primary": {
            "value": "convex-swath.delivery.puppetlabs.net"
        },
        "targets": {
            "type": "query",
            "value": "nodes[certname] { }"
        }
    }
}
```

You can use these keys in the `input` object for task jobs:

Key	Description
<code>name</code>	String-formatted name of the task to run.
<code>parameters</code>	An object that can be empty or contain key-value pairs representing standard parameter inputs for the task to use. Supply parameter name keys along with their corresponding values in a flat structure. The <code>value</code> key accepts any data type as long as it is valid JSON. For example: <pre>"parameters": { "param_one": "some_value" }</pre>

Key	Description
scope	<p>An object containing exactly one key defining the nodes that you want the task to target:</p> <ul style="list-style-type: none"> • <code>nodes</code>: A list of node names to target. • <code>query</code>: A PuppetDB query to use to generate a list of targeted nodes. • <code>node_group</code>: The ID of a classifier node group containing nodes to target. <p>For <code>scope</code> examples, refer to the request body examples in POST /command/deploy on page 681. Note that <code>application</code> is not a valid scope for a scheduled job.</p>
targets	An object containing connection information for SSH/WinRM targets.
noop	A Boolean specifying whether to run the task in no-op mode. The default is <code>false</code> .
concurrency	An integer specifying the maximum number of nodes to run at one time. The default, if unspecified, is unlimited.
transport	A string indicating the transport method over which to run the task.

Here is an example of a complete `input` object for a `task` job:

```
"input": {
    "name": "my_task",
    "scope": { "nodes": [ "my_primary" ] },
}
```

You can use these keys in the `input` object for `deploy` jobs:

Key	Description
scope	<p>Required. An object containing exactly one key defining the nodes that you want the deployment to target:</p> <ul style="list-style-type: none"> • <code>nodes</code>: A list of node names to target. • <code>query</code>: A PuppetDB query to use to generate a list of targeted nodes. • <code>node_group</code>: The ID of a classifier node group containing nodes to target. <p>For <code>scope</code> examples, refer to the request body examples in POST /command/deploy on page 681.</p>
concurrency	An integer specifying the maximum number of nodes to run at one time. The default, if unspecified, is unlimited.
debug	A Boolean specifying whether to use the <code>--debug</code> flag on Puppet agent runs.
enforce_environment	A Boolean specifying whether to force agents to run in the specified environment. This key must be <code>false</code> if <code>environment</code> is an empty string.
evaltrace	A Boolean specifying whether to use the <code>--evaltrace</code> flag on Puppet agent runs.
filetimeout	An integer specifying the value of the <code>--filetimeout</code> flag on Puppet agent runs.
http_connect_timeout	An integer specifying the value for the <code>--http_connect_timeout</code> flag on Puppet agent runs.
http_keepalive_timeout	An integer specifying the value for the <code>--http_keepalive_timeout</code> flag on Puppet agent runs.
http_read_timeout	An integer specifying the value for the <code>--http_read_timeout</code> flag on Puppet agent runs.
noop	A Boolean specifying whether to run the deployment in no-op mode. The default is <code>false</code> .
no_noop	A Boolean specifying whether to run the agent in enforcement mode. The default is <code>false</code> . This key overrides <code>noop = true</code> if set in the agent's <code>puppet.conf</code> file. This key can't be set to <code>true</code> if the <code>noop</code> key is also set to <code>true</code> in the same request.
ordering	A string defining the value of the <code>--ordering</code> flag on Puppet agent runs.

Key	Description
skip_tags	A string defining the value of the --skip_tags flag on Puppet agent runs.
tags	A string defining the value of the --tags flag on Puppet agent runs.
trace	A Boolean specifying whether to use the --trace flag on Puppet agent runs.
use_cached_catalog	A Boolean specifying whether to use the --use_cached_catalog flag on Puppet agent runs.
usecacheonfailure	A Boolean specifying whether to use the --usecacheonfailure flag on Puppet agent runs.

Here is an example of an input object for a deploy job:

```
"input": {
    "scope": { "nodes": [ "my_primary" ] },
}
```

Response format

If the job is successfully scheduled, the server returns 200 and a JSON object containing these keys:

- **id**: An absolute URL that links to the newly created job. You can use it with [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721 to retrieve information about the job.
- **name**: A stringified number identifying the newly created job.

For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs/environment_jobs/2",
    "name" : "2"
  }
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key.

Possible errors include:

Response code	Value	Definition
404	puppetlabs.orchestrator/unknown-environment	The specified environment does not exist.
400	puppetlabs.orchestrator/empty-environment	The specified environment contains no nodes.
400	puppetlabs.orchestrator/empty-target	The specified scope or targets query resolves to an empty list of nodes.

Response code	Value	Definition
400	puppetlabs/orchestrator/puppetdb-error	The request specified a query for the scope or targets, and the orchestrator is unable to make a query to PuppetDB.
400	puppetlabs/orchestrator/query-error	The request specified a query for the scope or targets, and the query is invalid or the user submitting the request does not have permission to run the query.
400	puppetlabs/orchestrator/invalid-time	The supplied start_time timestamp is in the past.

PUT /scheduled_jobs/environment_jobs/<job-id>

Edit or delete scheduled environment jobs.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the content type is application/json, the URI path must contain the ID of the job you want to edit, and the body must be a JSON object. The body can contain any of these top-level keys: input, environment, schedule, description, userdata, and enabled. Top-level keys specified in your request completely overwrite existing values for those keys. Omitted top-level keys are unchanged.



CAUTION:

If you supply a top-level key, the supplied value **completely replaces** the key's existing value for the scheduled job.

If you want to add new content to an existing value, such as adding a parameter to the input object, you must supply the key's entire current value *and* your additional new content.

For descriptions of keys and their contents, refer to [POST /scheduled_jobs/environment_jobs](#) on page 724. Pay particular attention to the input object, which can have a lot of nested values.

To make sure you aren't missing any values, you can use [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721 to get the job's current configuration.

While you can't truly delete a scheduled job, you can disable the job as a form of soft deletion. To disable a job, supply the following body in your request:

```
{
  "enabled": false
}
```

Response format

If the job is successfully edited, the server returns 200 and a JSON object containing the edited job's id and name. For example:

```
{
  "scheduled_job" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
scheduled_jobs/environment_jobs/2",
    "name" : "2"
  }
}
```

```
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the `kind` key. Possible errors include errors described on [POST /scheduled_jobs/environment_jobs](#) on page 724 and 400 `puppetlabs.orchestrator/disabled-scheduled-job` if you try to edit a disabled job (where `"enabled": false`).

GET /scheduled_jobs (deprecated)

Prior to deprecation, this endpoint retrieved information about scheduled jobs.

Important: This endpoint is deprecated. Instead, use [GET /scheduled_jobs/environment_jobs](#) on page 717 and [GET /scheduled_jobs/environment_jobs/<job-id>](#) on page 721. Tools using this deprecated endpoint must be upgraded to use the new endpoints.

While you can manually re-enable this endpoint, this is not recommended because this endpoint only returns an empty array. If you need to re-enable this endpoint, insert the following code in `orchestrator.conf`:

```
{
  orchestrator: {
    scheduled-jobs-v1-api: true;
  }
}
```

This also re-enables [DELETE /scheduled_jobs/<job-id> \(deprecated\)](#) on page 731.

Request format

Prior to deprecation, when [Forming orchestrator API requests](#) on page 678 to this endpoint, you could append parameters to the end of the URI path, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/scheduled_jobs?
limit=20&type=task
```

This endpoint accepted the same parameters as [GET /scheduled_jobs/environment_jobs](#) on page 717.

Response format

Because this endpoint is deprecated, the response is always a JSON object containing an empty `items` array.

```
{
  "items": []
}
```

DELETE /scheduled_jobs/<job-id> (deprecated)

Prior to deprecation, this endpoint deleted scheduled jobs.

Important: This endpoint is deprecated. Instead, use [PUT /scheduled_jobs/environment_jobs/<job-id>](#) on page 730. Tools using this deprecated endpoint must be upgraded to use the new endpoint.

While you can manually re-enable this endpoint, this is not recommended because this endpoint can't find jobs to delete. If you need to re-enable this endpoint, insert the following code in `orchestrator.conf`:

```
{
  orchestrator: {
    scheduled-jobs-v1-api: true;
  }
}
```

```
}
```

This also re-enables [GET /scheduled_jobs \(deprecated\)](#) on page 731.

Request format

Prior to deprecation, when [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must have included the ID of the scheduled job you wanted to delete. For example, this request deleted a scheduled job with ID 81:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/
scheduled_jobs/81"

curl --insecure --header "$auth_header" -X DELETE "$uri"
```

Response format

Because this endpoint is deprecated, the response is always 204 No Content.

Plans endpoints

Use the `plans` endpoints to get information about plans.

GET /plans

Lists all known plans in a specific environment.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can use the default URI path to query plans in the production environment, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/plans
```

For any other environments, you must use the `environment` parameter to specify the environment whose plans you want to query. For example, this request queries plans in the development environment:

```
https://orchestrator.example.com:8143/orchestrator/v1/plans?
environment=development
```

Response format

The response is a JSON object containing an `environment` object and a `items` array.

The `environment` object contains these keys:

- `name`: The environment specified in the request.
- `code_id`: Either null or a unique string. Puppet Server uses the `code_id` to retrieve the version of file resources in an environment at the time when a catalog was compiled. You can learn more about `code_id` in the [Puppet Static catalogs](#) documentation.

The `items` array contains one JSON object for each plan in the environment. Each plan object uses these keys:

Key	Definition
<code>id</code>	A URI path you can use with the GET /plans/<module>/<plan-name> on page 734 endpoint to learn more about the plan.

Key	Definition
name	The plan's name. You can use this with, for example, the POST /command/plan_run on page 695 endpoint.
permitted	A Boolean indicating if you are permitted to run the plan.

For example, this response describes three plans in the production environment:

```
{
  "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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs.orchestrator/validation-error	The environment parameter does not supply a legal environment name. For example, the name is not a string or contains illegal characters.
404	puppetlabs.orchestrator/unknown-environment	No environment exists that matches the specified environment.

GET /plans/<module>/<plan-name>

Get information about a specific plan, including metadata. This endpoint provides more information than the GET /plans endpoint.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include a specific module and plan name, such as:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/plans/profile/firewall"
```

Use the [GET /plans](#) on page 732 endpoint to get module and plan names.

If a plan is available in multiple environments, you can append the environment parameter to retrieve details about the plan in a specific environment. If you do not specify this parameter, the endpoint uses the default value, which is production. For example, this request retrieves details about the `firewall` plan in the development environment:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/plans/profile/firewall?environment=development"
```

Response format

The response is a JSON object that uses these keys to provide information about the specified plan:

Key	Definition
<code>id</code>	The URI path identifying the module and plan, as supplied in the request.
<code>name</code>	The plan's name. You can use this with, for example, the POST /command/plan_run on page 695 endpoint.
<code>environment</code>	A JSON object containing the name of the environment specified in the request and the <code>code_id</code> . The <code>code_id</code> is either <code>null</code> or a unique string that Puppet Server uses to retrieve the version of file resources in an environment at the time when a catalog was compiled. You can learn more about <code>code_id</code> in the Puppet Static catalogs documentation.

Key	Definition
metadata	<p>A JSON object that can be empty or contain a description of the plan and a parameters JSON object.</p> <p>The parameters object, if present, can use these keys:</p> <ul style="list-style-type: none"> • <code>type</code>: The required parameter type, as a valid Puppet type. If the parameter has no defined type, the value is Any. • <code>default_value</code>: The parameter's default value. Omitted if the parameter has no default value. • <code>description</code>: The parameter's description. Omitted if the parameter has no description. <p>Tip: The parameters are determined by the plan's Parameters key on page 670.</p>
permitted	A Boolean indicating if you are permitted to run the plan.

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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	There is a problem with the format of the module name, plan name, or environment parameter in the request. For example, one of the values contains illegal characters.

Response code	Key	Description
404	puppetlabs.orchestrator/unknown-environment	No environment exists that matches the specified environment.
404	puppetlabs.orchestrator/unknown-plan	<p>The endpoint can't find a match for the specified plan. There are several possible reasons for this, including:</p> <ul style="list-style-type: none"> • The endpoint can't find a valid relationship between the specified module and plan. • The plan name is well-formed but doesn't match any existing plans. • There is no such plan in the specified environment. <p>For example, a plan that only exists in the development environment returns 404 if the request specified a different environment or used the default environment value.</p>

Plan jobs endpoints

Use the `plan_jobs` endpoints to examine plan jobs and their details.

You can:

- [GET /plan_jobs](#) on page 736: Retrieve details about all known plan jobs.
- [GET /plan_jobs/<job-id>](#) on page 740: Retrieve details about a specific plan job.
- [GET /plan_jobs/<job-id>/events](#) on page 743: Retrieve a list of events that occurred during a specific plan job.
- [GET /plan_jobs/<job-id>/event/<event-id>](#) on page 746: Retrieve the details of a specific event for a specific plan job.

For details about jobs that aren't plan jobs, use the [Jobs endpoints](#) on page 703.

To stop an in-progress plan job, use [POST /command/stop_plan](#) on page 686.

GET /plan_jobs

Retrieve details about all plan jobs that the orchestrator knows about.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can append parameters to the end of the URI path, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs?
limit=20&order=desc
```

These parameters are available:

Parameter	Definition
limit	Set the maximum number of plan jobs to include in the response. The point at which the limit count starts is determined by offset, and the job record sort order is determined by order_by and order.
offset	Specify a zero-indexed integer at which to start returning results. For example, if you set this to 12, the response returns jobs starting with the 13th record. The default is 0.
order_by	Specify one of the following categories to use to sort the results: owner, timestamp, environment, name, or state. Sorting by owner uses the login subfield of owner records.
order	Indicate whether results are returned in ascending (asc) or descending (desc) order. The default is asc.
results	Whether you want the response to include or exclude plan output. The default is include.
min_finish_timestamp	Returns only the plan jobs that finished at or after the supplied UTC timestamp.
max_finish_timestamp	Returns only the plan jobs that finished at or before the supplied UTC timestamp.

Response format

The response is a JSON object containing an array, called items, and an object, called pagination.

The items array contains a JSON object for each plan job. Each object uses these keys to provide plan job details:

Key	Definition
id	The plan job's absolute URL, which includes the plan job's ID.
name	A stringified number identifying the plan job.
state	The plan job's current state: running, success, or failure
Tip: If you want to know when a plan job entered and exited each state, use the GET /plan_jobs/<job-id> on page 740 endpoint.	

Key	Definition
options	A JSON object containing plan job options, including: <ul style="list-style-type: none"> • description: A user-provided description of the plan job. • plan_name: The name of the plan that ran. • parameters: Parameters supplied to the plan job, such as target nodes. • scheduled_job_id: A job ID, if the plan ran as a scheduled job. Otherwise, the value is <code>null</code>. • environment: The environment the plan ran in. Omitted if not applicable. • sensitive: Password or SSH details supplied to the plan. Empty if not applicable. • project: Project information, such as a <code>project_id</code> and <code>ref</code>. Omitted if not applicable.
result	Plan output resulting from running the plan job. Omitted if you supplied <code>results=exclude</code> in the request.
owner	The subject ID, login, and other details of the user that requested the plan job.
created_timestamp	The time the plan job was created.
finished_timestamp	The time the plan job finished, or <code>null</code> if the job is currently running.
duration	If the plan job is finished, this is the number of seconds the job took to run. If the plan job is still running, this is the number of seconds the job has been running.
events	A link to the events associated with a plan job. You can use this with the GET /plan_jobs/<job-id>/events on page 743 endpoint.
userdata	An object of arbitrary key/value data supplied to the job.

The `pagination` object uses these keys:

- **total**: The total number of job records in the collection, regardless of `limit` and `offset`.
- **limit** and **offset**: Reflects values supplied in the request. If you specified a value, these key shows the value you specified. If you did not specify a value, the key shows the default value.

Here is an example response describing two plan jobs and pagination information:

```
{
  "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/orchestrator/v1/
plan_jobs/37/events"
  },
  "state": "running",
  "id": "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/37",
  "created_timestamp": "2018-06-06T20:22:08Z",
  "duration": 123.456,
  "options": {
    "description": "Testing myplan",
    "plan_name": "myplan",
    "parameters": {
      "nodes": [
        "orchestrator.example.com"
      ]
    }
  }
}
```

```

        ],
    },
    "sensitive": ["secret"],
    "environment": "production",
    "scheduled_job_id": 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
}
}
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 400 `puppetlabs/orchestrator/validation-error` response if there is a problem with a supplied parameter, such as the `limit` parameter not being formatted as an integer.

GET /plan_jobs/<job-id>

Retrieve details of a specific plan job, including the start and end times for each job state.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific plan job. Plan job IDs are returned in responses from plan-related [Command endpoints](#) on page 681 and the [GET /plan_jobs](#) on page 736 endpoint. For example, this request queries a plan job with ID 375:

```
https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/375
```

Job IDs are returned in responses from [Command endpoints](#) on page 681 and the [GET /jobs](#) on page 703 endpoint.

A complete request might look like:

```

auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8143/orchestrator/v1/
plan_jobs/81"

curl --insecure --header "$auth_header" "$uri"

```

Response format

The response is a JSON object that uses these keys to provide plan job details:

Key	Definition
<code>id</code>	The plan job's absolute URL, which includes the plan job's ID.
<code>name</code>	A stringified number identifying the plan job.
<code>state</code>	The plan job's current state: <code>running</code> , <code>success</code> , or <code>failure</code>
<code>options</code>	A JSON object containing plan job options, including: <ul style="list-style-type: none"> <code>description</code>: A user-provided description of the plan job. <code>plan_name</code>: The name of the plan that ran. <code>parameters</code>: Parameters supplied to the plan job, such as target nodes. <code>scheduled_job_id</code>: A job ID, if the plan ran as a scheduled job. Otherwise, the value is <code>null</code>. <code>environment</code>: The environment the plan ran in. Omitted if not applicable. <code>sensitive</code>: Password or SSH details supplied to the plan. Empty if not applicable. <code>project</code>: Project information, such as a <code>project_id</code> and <code>ref</code>. Omitted if not applicable.
<code>result</code>	Plan output resulting from running the plan job.
<code>owner</code>	The subject ID, login, and other details of the user that requested the plan job.
<code>timestamp</code>	The time when the plan job's <code>state</code> last changed.
<code>events</code>	A link to the events associated with the plan job. You can use this with the GET /plan_jobs/<job-id>/events on page 743 endpoint.

Key	Definition
status	A JSON object representing all jobs that ran as part of the plan. For each job, there is an array detailing each state the job was in while it ran, as well as the start and end times for each state.
userdata	Job states are different from plan job states. Job states include new, ready, running, stopping, stopped, finished, and failed. An object of arbitrary key/value data supplied to the job.

In this example response, two jobs ran as part of the plan, and each job had two states:

```
{
  "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-DDT18:45:31Z",
        "exit_time": null
      }
    ]
  }
}
```

```

        "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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The job ID in the request is not an integer.
404	puppetlabs/orchestrator/unknown-job	No plan job exists that matches the specified job ID.

GET /plan_jobs/<job-id>/events

Retrieve a list of events that occurred during a specific plan job.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer job ID identifying a specific plan job. Plan job IDs are returned in responses from plan-related [Command endpoints](#) on page 681 and the [GET /plan_jobs](#) on page 736 endpoint.

You can use the optional start parameter to start the list of events from a specific event ID number.

For example, this request queries events associated with the 352 plan job, starting with event number 1272:

```
GET https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/
events?start=1272
```

Response format

A successful response is a JSON object containing a next-events object and an items array.

The next-events object contains two subkeys:

- id: The URI supplied in the request.
- event: The ID of the first event returned or the start parameter, if supplied in the request.

The items array uses these keys to detail the plan job's events:

Key	Definition
id	An individual event's ID

Key	Definition
type	<p>Each event has one event type, determined by the event's status or circumstances that cause it to occur:</p> <ul style="list-style-type: none"> • <code>task_start</code>: A task run started. • <code>script_start</code>: A script run starts as part of a plan. • <code>command_start</code>: A command run starts as part of a plan. • <code>upload_start</code>: A file upload starts as part of a plan. • <code>wait_start</code>: A <code>wait_until_available()</code> call starts as part of a plan. • <code>out_message</code>: As part of a plan, <code>out::message</code> is called. • <code>apply_start</code>: A <code>puppet apply</code> run started as part of a plan. • <code>apply_prep_start</code>: An <code>apply_prep</code> run starts as part of a plan. • <code>plan_finished</code>: The plan job successfully finished. • <code>plan_failed</code>: The plan job failed. <p>A plan containing the <code>run_plan()</code> function completes the secondary plan during the primary plan job. Such <i>subplans</i> do not have their own plan jobs – They are included with, and completed as part of, the original job. These event types indicate when a subplan started or finished:</p> <ul style="list-style-type: none"> • <code>plan_start</code>: The <code>run_plan()</code> function started a plan within the current plan job. • <code>plan_end</code>: The subplan, triggered by the <code>run_plan()</code> function, finished. This event type is specific to subplans and different from <code>plan_finished</code>.

Key	Definition
timestamp	The time the event occurred or was created.
details	<p>A JSON object containing information about the event. Specific contents depends on the event type:</p> <ul style="list-style-type: none"> For any *_start events (except plan_start), details include the job-id of the associated action or task. For plan_finished and plan_failed events, details include the plan-id and result. For out_message events, details include the message contents, truncated to 1,024 bytes. You can use the GET /plan_jobs/<job-id>/event/<event-id> on page 746 endpoint to get the full message content. For plan_start events, details include the plan, which identifies the subplan that ran within the original plan. For plan_end events, details include the plan (which is the subplan that ran within the original plan) and the duration (which is how long the subplan ran).

Here is an example response body:

```
{
  "next-events" : {
    "id" : "https://orchestrator.example.com:8143/orchestrator/v1/
plan_jobs/352/events?start=1272",
    "event": "1272"
  },
  "items" : [ {
    "id" : "1273",
    "type" : "task_start",
    "timestamp" : "2016-05-05T19:50:08Z",
    "details" : {
      "job-id" : "8765"
    }
  },
  {
    "id" : "1274",
    "type" : "plan_finished",
  }
]
```

```

    "timestamp" : "2016-05-05T19:50:14Z",
    "details" : [
      "plan-id" : "1234",
      "result" : [
        "Plan output"
      ],
    ],
  ],
}

```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The plan job ID or the start parameter in the request are not supplied as integers.
404	puppetlabs/orchestrator/unknown-job	No plan job exists that matches the specified plan job ID.

GET /plan_jobs/<job-id>/event/<event-id>

Retrieve the details of a specific event for a specific plan job.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer plan job ID and an integer event ID. You can get plan job IDs from plan-related [Command endpoints](#) on page 681 and the [GET /plan_jobs](#) on page 736 endpoint. You can get event IDs from the [GET /plan_jobs/<job-id>/events](#) on page 743 endpoint.

For example, this request queries event number 1272 for plan job number 352:

```
GET https://orchestrator.example.com:8143/orchestrator/v1/plan_jobs/352/
event/1272
```

Tip: The URI path uses the singular event, and not events, like the [GET /plan_jobs/<job-id>/events](#) on page 743 endpoint.

Response format

A successful response is a JSON object that uses these keys to provide the event details:

Key	Definition
id	The event's ID.

Key	Definition
type	<p>The event's type, determined by the event's status or circumstances that cause it to occur:</p> <ul style="list-style-type: none"> • <code>task_start</code>: A task run started. • <code>script_start</code>: A script run starts as part of a plan. • <code>command_start</code>: A command run starts as part of a plan. • <code>upload_start</code>: A file upload starts as part of a plan. • <code>wait_start</code>: A <code>wait_until_available()</code> call starts as part of a plan. • <code>out_message</code>: As part of a plan, <code>out::message</code> is called. • <code>apply_start</code>: A <code>puppet apply</code> run started as part of a plan. • <code>apply_prep_start</code>: An <code>apply_prep</code> run starts as part of a plan. • <code>plan_finished</code>: The plan job successfully finished. • <code>plan_failed</code>: The plan job failed. <p>A plan containing the <code>run_plan()</code> function completes the secondary plan during the primary plan job. Such <i>subplans</i> do not have their own plan jobs – They are included with, and completed as part of, the original job. These event types indicate when a subplan started or finished:</p> <ul style="list-style-type: none"> • <code>plan_start</code>: The <code>run_plan()</code> function started a plan within the current plan job. • <code>plan_end</code>: The subplan, triggered by the <code>run_plan()</code> function, finished. This event type is specific to subplans and different from <code>plan_finished</code>.

Key	Definition
timestamp	The time the event occurred (or was created).
details	<p>A JSON object containing information about the event. Specific contents depends on the event type:</p> <ul style="list-style-type: none"> For any *_start events (except plan_start), details include the job-id of the associated action or task. For plan_finished and plan_failed events, details include the plan-id and result. For out_message events, details include the complete message contents. For plan_start events, details include the plan, which identifies the subplan that ran within the original plan. For plan_end events, detail include the plan (which is the subplan that ran within the original plan) and the duration (which is how long the subplan ran).

Here is an example of a response body:

```
{
  "id": "1265",
  "type": "out_message",
  "timestamp": "2016-05-05T19:50:06Z",
  "details": {
    "message": "this is an output message"
  }
}
```

Error responses

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The plan job ID or the event ID in the request are not supplied as integers.
404	puppetlabs/orchestrator/unknown-job	No plan job exists that matches the specified plan job ID.

Response code	Key	Description
404	puppetlabs.orchestrator/mismatched-job-event-id	The specified event ID does not match any event ID associated with the specified plan job ID.

Tasks endpoints

Use the `tasks` endpoints to get information about tasks you've installed and tasks included with Puppet Enterprise (PE).

GET /tasks

Lists all tasks in a specific environment.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can use the default URI path to query tasks in the production environment, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/tasks
```

For any other environments, you must use the `environment` parameter to specify the environment whose tasks you want to query. For example, this request queries tasks in the development environment:

```
https://orchestrator.example.com:8143/orchestrator/v1/plans?environment=development
```

Response format

The response is a JSON object containing an `environment` object and a `items` array.

The `environment` object contains these keys:

- `name`: The environment specified in the request.
- `code_id`: Either null or a unique string specifying where the environment's tasks are listed. Puppet Server uses the `code_id` to retrieve the version of file resources in an environment at the time when a catalog was compiled. You can learn more about `code_id` in the Puppet [Static catalogs](#) documentation.

The `items` array contains one JSON object for each task in the environment. Each task object uses these keys:

Key	Definition
<code>id</code>	A URI path you can use with the GET /tasks/<module>/<task-name> on page 750 endpoint to learn more about the task.
<code>name</code>	A stringified number identifying the task. You can use this with, for example, the POST /command/task on page 687 endpoint.

For example, this response describes three tasks in the production environment:

```
{
  "environment": {
    "name": "production",
    "code_id": "urn:puppet:code-
id:1:a86da166c30f871823f9b2ea224796e834840676;production"
  },
  "items": [
    {
      "id": "1:a86da166c30f871823f9b2ea224796e834840676;production"
    }
  ]
}
```

```

        "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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	The environment parameter does not supply a legal environment name. For example, the name is not a string or contains illegal characters.
404	puppetlabs/orchestrator/unknown-environment	No environment exists that matches the specified environment.

GET /tasks/<module>/<task-name>

Get information about a specific task, including metadata and file information. This endpoint provides more information than the GET /tasks endpoint.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include a specific module and task name, such as:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/
install"
```

Use the [GET /tasks](#) on page 749 endpoint to get module and plan names.

To request a module's default task, use init as the task name, such as:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/
init"
```

If a task is available in multiple environments, you can append the environment parameter to retrieve details about the task in a specific environment. If you do not specify this parameter, the endpoint uses the default value, which is production. For example, this request retrieves details about the install task in the development environment:

```
GET "https://orchestrator.example.com:8143/orchestrator/v1/tasks/package/
install?environment=development"
```

Response format

The response is a JSON object that uses these keys to provide information about the specified task:

Key	Definition
id	The URI path identifying the module and task, as supplied in the request.
name	A stringified number identifying the task. You can use this with, for example, the POST /command/task on page 687 endpoint.
environment	A JSON object containing the name of the environment specified in the request and the <code>code_id</code> . <code>code_id</code> is either <code>null</code> or a unique string specifying where the environment's tasks are listed. Puppet Server uses the <code>code_id</code> to retrieve the version of file resources in an environment at the time when a catalog was compiled. You can learn more about <code>code_id</code> in the Puppet Static catalogs documentation.
metadata	A JSON object containing the Task metadata on page 641.
files	An array of JSON objects describing files used by the task. Each file object can use these keys: <ul style="list-style-type: none"> • <code>filename</code>: The base name of the file. • <code>uri</code>: An object containing the path and <code>params</code> you can use to locate the file and the version of the file used (such as the version from the production environment). 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:a86da166c30f871823f9b2ea224796e834840676;production"
  },
  "metadata": {
    "description": "Install a package",
    "supports_noop": true,
    "input_method": "stdin",
    "parameters": {
      "name": {
        "description": "The package to install",
        "type": "string"
      }
    }
  }
}
```

```

        "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

If there is an error, [Orchestrator API error responses](#) on page 758 provide error information in the kind key:

Response code	Key	Description
400	puppetlabs/orchestrator/validation-error	There is a problem with the format of the module name, task name, or environment parameter in the request. For example, one of the values contains illegal characters.
404	puppetlabs/orchestrator/unknown-environment	No environment exists that matches the specified environment.

Response code	Key	Description
404	puppetlabs/orchestrator/unknown-task	<p>The endpoint can't find a match for the specified task. There are several possible reasons for this, including:</p> <ul style="list-style-type: none"> • The endpoint can't find a valid relationship between the specified module and task. • The task name is well-formed but doesn't match any existing tasks. • There is no such task in the specified environment. <p>For example, a task that only exists in the development environment returns 404 if the request specified a different environment or used the default environment value.</p>

Usage endpoints

Use the `usage` endpoint to view details about your deployment's active nodes.

For information about how nodes are counted, which nodes are counted, node usage limitations, and monthly busting limits, refer to [How nodes are counted](#) on page 437.

GET /usage

Retrieves information about the orchestrator's daily node usage, Puppet events activity on nodes, and nodes that are present in PuppetDB.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, you can append parameters to the end of the URI path, such as:

```
https://orchestrator.example.com:8143/orchestrator/v1/usage?
start_date=2022-01-01&end_date=2022_04_30&events=exclude
```

These parameters are available:

Parameter	Definition
<code>start_date</code>	Specify the earliest date to query, in YYYY-MM-DD format.
<code>end_date</code>	Specify the latest date to query, in YYYY-MM-DD format. If you also specified <code>start_date</code> , the <code>end_date</code> must be greater than or equal to the <code>start_date</code> .

Parameter	Definition
events	<p>Specifies whether you want the response to include or exclude daily Puppet events activity. The default is <code>include</code>.</p> <p>If set to <code>exclude</code>, the response only contains node counts (total nodes and the number of nodes with and without agents). Specifically, the response omits the number of corrective changes, the number of intentional changes, the number of task runs, and the number of plan runs.</p>

Response format

The response is a JSON object containing an array, called `items`, and an object, called `pagination`.

The `pagination` object contains the `start_date` and `end_date` parameters as supplied in the request.

The `items` array contains one JSON object for each day. Each object uses these keys to provide details about daily node usage:

Key	Definition
<code>date</code>	An ISO-8601 date representing the day in UTC.
<code>total_nodes</code>	The total number of nodes used on a particular date, starting from midnight UTC. Unused or inactive nodes are not counted.
<code>nodes_with_agent</code>	The number of unique nodes, out of the <code>total_nodes</code> , that have an agent installed. This is the number of nodes in PuppetDB on a particular date.
<code>nodes_without_agent</code>	The number of unique nodes, out of the <code>total_nodes</code> , that do not have an agent installed.
<code>corrective_agent_changes</code>	The number of corrective changes made by agent runs on a particular date. Omitted if the request contained <code>events=exclude</code> .
<code>intentional_agent_changes</code>	The number of intentional changes made by agent runs on a particular date. Omitted if the request contained <code>events=exclude</code> .
<code>nodes_affected_by_task_runs</code>	The number of tasks run (counted per node that a task runs on) on a particular date. Omitted if the request contained <code>events=exclude</code> .
<code>nodes_affected_by_plan_runs</code>	The number of plans run (counted per node that a plan runs on) on a particular date. Omitted if the request contained <code>events=exclude</code> .

For example, this is a response to a request that contained `events=exclude`:

```
{
  "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

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a `400 puppetlabs/orchestrator/validation-error` response if there is a problem with a supplied parameter, such as the `start_date` parameter not having the proper date format.

Scopes endpoints

Use the `scopes` endpoints to retrieve information about task-targets.

A *task-target* is a set of tasks and nodes/node groups you can use to provide specific privilege escalation for users who would otherwise not be able to run certain tasks or run tasks on certain nodes or node groups. When you grant a user permission to use a task-target, the user can run the task(s) in the task-target on the set of nodes defined in the task-target. Use the [POST /command/task_target](#) on page 691 endpoint to create task-targets.

GET /scopes/task_targets

Retrieve information about all orchestrator task-targets.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the request is a basic call, such as:

```
GET https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets
```

The GET `/scopes/task_targets` endpoint does not support any parameters; however, as with other orchestrator API endpoints, you must provide authentication.

Tip: The GET `/scopes/task_targets` endpoint returns information about all known task-targets. When there are many task-targets, the response body contains lots of data. For shorter responses, you can use the [GET /scopes/task_targets/<task-target-id>](#) on page 757 endpoint to query a specific task-target, if you know the task-target's ID.

Response format

A successful response is a JSON object containing an array of task-targets. The following keys are used to provide information about each task-target:

Key	Definition
<code>id</code>	The task-target's absolute URL, which includes the task-target's numerical ID.
<code>name</code>	A stringified number identifying the task-target.
<code>display_name</code>	The task-target's human-readable name. Multiple task-targets can have the same display name.
<code>tasks</code>	An array of tasks that the task-target can run. Omitted if <code>all_tasks</code> is <code>true</code> .
<code>all_tasks</code>	A Boolean indicating whether the task-target can run any tasks on designated node targets. If <code>tasks</code> is specified, then <code>all_tasks</code> is <code>false</code> . If <code>tasks</code> is omitted, then <code>all_tasks</code> is <code>true</code> .
<code>nodes</code>	An array of certnames identifying nodes the task-target can run tasks on. It can be empty. Combines with <code>node_groups</code> and <code>pql_query</code> to form a total node pool.
<code>node_groups</code>	An array of node group IDs identifying node groups the task-target can run tasks on. It can be empty. Combines with <code>nodes</code> and <code>pql_query</code> to form a total node pool.
<code>pql_query</code>	A string specifying a single PQL query identifying nodes the task-target can run tasks on. Omitted if empty. Combines with <code>nodes</code> and <code>node_groups</code> to form a total node pool.

Tip: For information about how these keys are set and possible values for each key, refer to the [POST /command/task_target](#) on page 691 endpoint.

For example, this response describes three task-targets:

```
{
  "items": [
    {
      "id": "https://orchestrator.example.com:8143/orchestrator/v1/scopes/task_targets/1",
      "name": "1",
      "tasks": [
        "package::install",
        "exec"
      ],
      "all_tasks": "false",
      "nodes": [
        "wss6c3w9wngpycg",
        "jjj2h5w8gpycgn"
      ],
      "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"
        ]
    }
]
}

```

Error responses

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758.

GET /scopes/task_targets/<task-target-id>

Get information about a specific task-target.

Request format

When [Forming orchestrator API requests](#) on page 678 to this endpoint, the URI path must include an integer identifying a specific task-target. For example, this request queries a task-target with ID 375:

```
https://orchestrator.example.com:8143/orchestrator/v1/scopes/
task_targets/375
```

Task-target IDs are returned in responses from the [POST /command/task_target](#) on page 691 and [GET /scopes/task_targets](#) on page 755 endpoints.

Response format

The response is a JSON object that uses the following keys to provide details about the task-target:

Key	Definition
id	The task-target's absolute URL, which includes the task-target's numerical ID.
name	A stringified number identifying the task-target.
display_name	The task-target's human-readable name. Multiple task-targets can have the same display name.

Key	Definition
tasks	An array of tasks that the task-target can run. Omitted if all_tasks is true.
all_tasks	A Boolean indicating whether the task-target can run any tasks on designated node targets. If tasks is specified, then all_tasks is false. If tasks is omitted, then all_tasks is true.
nodes	An array of certnames identifying nodes the task-target can run tasks on. It can be empty. Combines with node_groups and pql_query to form a total node pool.
node_groups	An array of node group IDs identifying node groups the task-target can run tasks on. It can be empty. Combines with nodes and pql_query to form a total node pool.
pql_query	A string specifying a single PQL query identifying nodes the task-target can run tasks on. Omitted if empty. Combines with nodes and node_groups to form a total node pool.

Tip: For information about how these keys are set and possible values for each key, refer to the [POST /command/task_target](#) on page 691 endpoint.

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

This endpoint's error responses follow the usual format for [Orchestrator API error responses](#) on page 758. The endpoint returns a 404 `puppetlabs/orchestrator/unknown-task-target` response if the specified task-target ID doesn't match any existing task-target IDs.

Orchestrator API error responses

Orchestrator API error responses are formatted as JSON objects.

Error responses use these keys:

Key	Definition
kind	The kind of error encountered.

Key	Definition
msg	The message associated with the error.
details	A hash with more information about the error.

For example, if an environment does not exist for a given request, you might get this error response:

```
{
  "kind" : "puppetlabs.orchestrator/unknown-environment",
  "msg" : "Unknown environment doesnotexist",
  "details" : {
    "environment" : "doesnotexist"
  }
}
```

Migrating Bolt tasks and plans to PE

If you use Bolt tasks and plans to automate parts of your configuration management, you can move that Bolt content to a control repo and transform it into a Puppet Enterprise (PE) environment. This lets you manage and run tasks and plans using PE and the console. Bolt projects have the same structure as Puppet modules, and they can be loaded from the `modules` directory of a PE environment.

The *control repo* is a central Git repository from which PE fetches content. An *environment* is a space for PE authors to write and install content, similar to a Bolt project.

There are two ways to get your Bolt content into an environment:

- Move your Bolt code to a new control repo. Do this if you have a `Boltdir`, or an [embedded project directory](#), in a repo that also contains other code that you do not want to migrate to PE.
- Configure PE to point to the Bolt project. Do this 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 65.
- Set up your PE control repo and environments. See [Managing environments with a control repository](#) on page 763.

To move Bolt content to a repo:

- Commit the contents of your Bolt project to a branch of your PE control repo. Place the Bolt project under the `modules` directory. If you're using Bolt [module workflows](#), make sure you run `bolt module install` and commit the resulting Puppetfile to your control repo.

Your new structure is similar to a [project directory](#) in Bolt, for example:

```
test-environment/
  ### Puppetfile
  ### bolt-project.yaml
  ### data
  #   ### common.yaml
  ### inventory.yaml
  ### modules
    ### project
      ### manifests
      #       ### my_class.pp
      ### plans
      #       ### deploy.pp
      #       ### diagnose.pp
      ### tasks
        ### init.json
        ### init.py
```

- Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment in PE.
- Add the `modulepath` setting to the `environment.conf` file by adding the following line:

```
modulepath = modules:modules:$basemodulepath
```

Note: PE picks up modules only from the `modules` directory. It's important to add `modules` to the `modulepath` setting so it matches the defaults for your Bolt project. If you have a `modulepath` setting in `bolt-project.yaml`, match it to the `modulepath` setting in `environment.conf`.

- Publish the branch to the PE control repo.
- 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 801 for more information.

After you deploy code, modules (and the tasks and plans within them) listed in the new environment's Puppetfile are available to use in PE.

Related information

[Plans in PE versus Bolt plans](#) on page 646

Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

Point PE to a Bolt project

Allow PE to manage content in your dedicated Bolt repo.

Before you begin

- Install PE on your machine. See [Getting started with Puppet Enterprise](#) on page 65.
- Ensure your Bolt project follows the [local project directory](#) structure.

To point PE to your Bolt content:

- To allow access to the control repo, generate a private SSH key without a password:

- To generate the key pair, run:

```
ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

- To allow the pe-puppet user to access the key, run:

```
puppet infrastructure configure
```

Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.

- Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as [GitHub](#), [BitBucket Server](#), or [GitLab](#)).

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

- Change the name of your branch to `production`. PE uses branches in Git as environments and the default environment is `production`.
- Create a configuration file called `environment.conf` and add it to the root directory of the branch. This file configures the environment.
- Add the `modulepath` setting to the `environment.conf` file by adding the following line:

```
modulepath = modules:modules:$basemodulepath
```

Note: PE picks up modules only from the `modules` directory. It's important to add `modules` to the `modulepath` setting so it matches the defaults for your Bolt project. If you have a `modulepath` setting in `bolt-project.yaml`, match it to the `modulepath` setting in `environment.conf`.

- Publish the branch to the PE control repo.
- 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 801 for more information.

After you deploy code, modules (and the tasks and plans within them) listed in the new environment's Puppetfile are available to use in PE.

Related information

[Plans in PE versus Bolt plans](#) on page 646

Some plan language functions, features, and behaviors are different in PE than they are in Bolt. If you are used to Bolt plans, familiarize yourself with some of these key differences and limitations before you attempt to write or run plans in PE.

PE workflows for Bolt users

Understand the differences between PE and Bolt commands and workflows before you start running tasks and plans in PE.

Connecting to nodes

You must connect PE to each node you want to run tasks on or include in a plan. See [Add nodes to the inventory](#) on page 73 for instructions on adding agent or agentless nodes to your inventory.

Installing tasks and plans

In PE, as in Bolt, you use the `mod` command to download modules. But instead of running the `bolt puppetfile install` command to install them, you trigger Code Manager and deploy code using the `puppet code deploy` command. See [Triggering Code Manager on the command line](#) on page 795.

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 622
- [Running plans from the command line](#) on page 650

To run tasks or plans from the console, see:

- [Running tasks from the console](#) on page 616
- [Running plans from the console](#) on page 649

Limitations in PE

Not everything in Bolt works in PE. For example, many pre-installed Bolt modules are not included in PE and many plan functions do not work, such as `file::exists` and `set_feature`. See [Plans in PE versus Bolt plans](#) on page 646.

Managing and deploying Puppet code

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

You can use either Code Manager or r10k to manage and deploy Puppet code. Both tools are built into PE and do not require separate installations.

Code Manager is the recommended tool for managing Puppet code in PE. Code Manager automates the deployment of your Puppet code and data. You make code and data changes on your workstation, push changes to your Git repository, and, from there, Code Manager creates environments, installs modules, and deploys the new code to your primary server and compilers, without interrupting agent runs.

If you are unable to use Code Manager, you can use r10k to manage your code.

- [Managing environments with a control repository](#) on page 763

To manage your Puppet code and data with Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

- [Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

- [Managing code with Code Manager](#) on page 774

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

- [Managing code with r10k](#) on page 821

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to *manually* deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

Managing environments with a control repository

To manage your Puppet code and data with Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

How the control repository works

Code management relies on version control to track, maintain, and deploy your Puppet code and data. The control repository (or control repo) is a Git repository that code management uses to manage environments in your infrastructure. When you update code and data in your control repo, code management updates your environments accordingly.

Code management creates and maintains environments based on the branches in your control repo. For example, if your control repo has a `production` branch, a `development` branch, and a `testing` branch, code management creates a `production` environment, a `development` environment, and a `testing` environment. Each environment has its own version of your Puppet code and data based on the contents of the corresponding branch.

Environments are created on the primary server at `/etc/puppetlabs/code/environments`. You can learn more [About environments](#) in the Puppet documentation.



CAUTION: When you enable code management, Puppet manages the environment directories and **does not preserve existing environments**. Existing environments with the same names as new ones are overwritten, and environments not represented in the control repo are erased. If you were already using environment directories, make sure you commit those files or code to the corresponding branches of your control repo (or back them up elsewhere) **before** you start configuring code management.

At minimum, a control repo includes:

- A Git remote repository. This is where your control repo is stored on your version control host.
- A default branch named `production`, rather than the usual Git default of `master`. You might have additional branches for other environments, such as `development` or `testing`.
- A `Puppetfile` to manage your environment content.
- An `environment.conf` file that modifies the `$modulepath` setting to allow environment-specific modules and settings.

There are two ways to create control repos. To ensure your control repo has the recommended structure, code examples, and configuration scripts, [Create a control repository from the Puppet template](#) on page 74. This template covers most customer situations. If you cannot access the internet or cannot download modules directly from the Forge because of your organization's security rules, [Create an empty control repo](#) on page 765 and add the necessary files to it.

Restriction: For Windows systems, make sure your version control is configured to use CRLF line endings. Check your version control host's documentation for instructions on how to do this.

It is possible to have multiple control repos, and you can use separate repos to contain your module content or other data. If you have multiple repos, you need to:

- Configure the `repositories` setting in your code management tool's Git settings:
 - [Configuring Code Manager Git settings](#)
 - [Configuring r10k Git settings](#)
- Map your sources.
 - [Configuring Code Manager sources](#)
 - [Configuring r10k sources](#)

Create a control repository from the Puppet template

To create a control repository (or control repo) that has the recommended structure, code examples, and configuration scripts, base your control repo on the Puppet control repo template. This template covers most customer situations.

The [Puppet control repo template](#) contains the necessary files to configure a functioning code management control repo plus helpful Puppet code examples, including:

- Basic code examples for setting up roles and profiles.
- A Puppetfile that references modules to manage content in your environments.
- An example Hiera configuration file and `hieradata` directory.
- A `config_version` script that tells you which version of code from your control repo was applied to your agents.
- An `environment.conf` file that implements the `config_version` script and a `site-modules` directory for roles, profiles, and custom modules.

In situations where you can't access the internet, or where organizational security policies prevent downloading modules from the Forge, you can [Create an empty control repo](#) on page 765 and add the necessary files to it.

To use the template, you must set up a private SSH key, copy the control repo template to your development workstation, set your own remote Git repository as the default source, and then push the template contents to that source.

Important: The following steps assume you are using GitHub Enterprise with SSH. For more information and instructions for other version control hosts, such as GitLab or BitBucket, go to the [Puppet control-repo template README](#).

1. To allow access to the control repo, generate a private SSH key without a password:

- a) To generate the key pair, run:

```
ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

- b) To allow the `pe-puppet` user to access the key, run:

```
puppet infrastructure configure
```

Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.

- c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as [GitHub](#), [BitBucket Server](#), or [GitLab](#)).

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

2. In your Git user account or organization, create a repository named `control-repo`, and make sure a README is **not** automatically generated when you create the repo. Take note of the repo's SSH URL.

Important: Do not use an existing repo. The template requires a new, empty repo named `control-repo`.

3. If you haven't already installed Git, run `yum install git`.
4. To clone the Puppet control-repo template, run:

```
git clone https://github.com/puppetlabs/control-repo.git
```

5. Change to the `control-repo` directory: `cd control-repo`
6. Remove the template repo as the origin: `git remote remove origin`

7. Set your control repo as the origin: `git remote add origin <SSH_URL_FOR_YOUR_CONTROL_REPO>`
8. Push the contents of the production branch of the cloned control repo to your remote control repo: `git push origin production`

You now have a control repository based on the Puppet control-repo template. After configuring Code Manager, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, Code Manager detects and deploys your infrastructure changes.

By using the control-repo template, you now also have a Puppetfile to which you can add and manage content, like module code.

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

[Managing code with Code Manager](#) on page 774

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

[Add an environment](#) on page 767

Create new environments by creating branches based on your control repo's production branch.

Create an empty control repo

In situations where you can't access the internet, or where organizational security policies prevent downloading modules from the Forge, you can create an empty control repo and add the necessary files to it.

When you can't use the Puppet control repo template, you must create a new repo on your Git host, clone it to your workstation, make changes to the repo (such as adding a configuration file to allow code management tools to find modules in your module directories), and push your changes to the remote repo on your Git host.

1. To allow access to the control repo, generate a private SSH key without a password:

- a) To generate the key pair, run:

```
ssh-keygen -t ed25519 -P '' -f /etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

- b) To allow the pe-puppet user to access the key, run:

```
puppet infrastructure configure
```

Your private key is located at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`, and your public key is at `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519.pub`.

- c) Configure your Git host to use the SSH public key you generated. Usually, this involves creating a user or service account and assigning the SSH public key to it, but the exact process varies for each Git host. For instructions on adding SSH keys to your Git server, check your Git host's documentation (such as [GitHub](#), [BitBucket Server](#), or [GitLab](#)).

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

2. In your Git account, create a repository with the name you want your control repo to have (we recommend `control-repo`), and take note of the repo's SSH URL.

Check your Git host's documentation for exact instructions, because this process varies for each host. For example, to create a new repo on GitHub:

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

Tip: While you can use an existing repo as your control repo, we recommend starting with a new repo to avoid possible unexpected changes to existing files and directories once you enable code management.

3. Clone the new repo to your workstation: `git clone <REPOSITORY_URL>`

4. In the control repo's main directory, create a configuration file named `environment`.

The `environment.conf` file allows code management tools to find modules in your site- and environment-specific module directories. You can learn more about this file and its contents in the Puppet [environment.conf](#) documentation.

5. To set the module path, open the `environment.conf` file in a text editor, add the following line, and then save and close the file.

```
modulepath=site-modules:modules:$basemodulepath
```

6. Add the new file to the index and commit your change by running `git add environment.conf` and then `git commit -m "add environment.conf"`

7. Rename the master branch to production by running `git branch -m master production`

Important: Puppet Enterprise requires the control repo's default branch to be `production`.

8. Push your repository's `production` branch from your workstation to your Git host by running: `git push -u origin production`

After configuring the Puppetfile and code management, when you make changes to your control repo on your workstation and push the changes to the remote control repo on your Git host, code management detects and deploys your infrastructure changes.

After creating your control repo, you must create a Puppetfile to manage your environment content with code management. Then, you must configure either Code Manager (recommended) or r10k.

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

[Managing code with Code Manager](#) on page 774

Code Manager automates the management and deployment of your Puppet code. When you push code updates to your source control repository, Code Manager syncs the code to your primary server and compilers. This allows all your servers to run the new code as soon as possible, without interrupting in-progress agent runs.

[Managing code with r10k](#) on page 821

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to *manually* deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

[Add an environment](#) on page 767

Create new environments by creating branches based on your control repo's production branch.

Add an environment

Create new environments by creating branches based on your control repo's production branch.

Before you begin

You must have:

- A [control repository](#).
- A [Puppetfile](#) in your control repo's production branch.
- Configured [Code Manager](#) or [r10k](#).
- Selected a code management deployment method (either the [puppet-code](#) command or a [webhook](#)).

Restriction: If you have multiple control repos, you can't repeat branch names unless you use a source prefix. Go to [Configuring sources](#) on page 790 for more information.

1. In your control repo, create a new branch based on the production branch: `git branch <NEW_BRANCH_NAME>`
2. Check out the new branch: `git checkout <NEW_BRANCH_NAME>`
3. Edit the Puppetfile to track the necessary modules and data for your new environment, and then save your changes.
4. Commit your changes: `git commit -m "prepare Puppetfile for new environment"`
5. Push your changes: `git push origin <NEW_BRANCH_NAME>`
6. Deploy your environments as you normally would, either on the command line or with a webhook.

Code management detects the new environment in your control repo and begins managing it, as explained in [How the control repository works](#) on page 763.

Delete an environment from code management

To delete an environment that is being managed by Code Manager or r10k, delete the corresponding branch from your control repository.

1. On your control repo's production branch, delete the environment's corresponding remote branch by running `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 webhook.

Important: If you use webhooks to deploy environments, Code Manager deletes the environment from the primary server's live code directories the next time it deploys changes to any other environment. If you want to immediately delete the environment from the primary server's live code directories, deploy all environments manually by running `puppet-code deploy --all --wait`

Managing environment content with a Puppetfile

A Puppetfile specifies detailed information about each environment's Puppet code and data.

The Puppetfile also specifies where to locate each environment's Puppet code and data, where to install it, and whether to update it. Both Code Manager and r10k use a Puppetfile to install and manage your environments' content.

The Puppetfile

Your control repository's branches represent environments, and each environment might have different modules or data. To manage each environment's content, you need a Puppetfile. In the Puppetfile, you specify which modules and data you want in each environment.

A Puppetfile is a formatted text file that specifies the modules and data you want in your control repository (where each branch of the control repo represents an environment). The Puppetfile can specify desired module versions, how to load modules and data, and where to place modules and data in the environment. In your Puppetfile you can declare:

- Modules from the [Forge](#).
- Modules from Git repositories.
- Data and other non-module content (such as Hiera data) from Git repositories.

You can declare as much or as little of this content as needed for each environment. In the Puppetfile, each module or repository is specified by a mod directive, along with the name of the content and other information code management needs to correctly install and maintain the declared modules and data.

Related information

[Managing environments with a control repository](#) on page 763

To manage your Puppet code and data with Code Manager or r10k, you need a Git version control repository. This control repository is where code management stores code and data to deploy your environments.

[Create a Puppetfile](#) on page 769

The Puppetfile manages an environment's content. When you create a Puppetfile, use the mod directive to declare an environment's content.

Managing modules with a Puppetfile

Almost all Puppet manifests are kept in *modules*, which are collections of Puppet code and data that have a specific directory structure. With Puppet Enterprise (PE) code management, you only use the Puppetfile to install and manage modules.

To learn more about modules in general, refer to the [Modules overview](#) in the Puppet documentation.

By default, Code Manager and r10k install module content in a modules directory in the same directory the Puppetfile is in. For example, with the default settings, declaring the `puppetlabs-apache` module in your Puppetfile installs the `apache` module into the `./modules/apache` directory. However, you can [Change the module installation directory](#) on page 774.

Important: Code management purges any content in your control repo's `module` directory that **is not** listed in your Puppetfile. For this reason, if you use Code Manager or r10k, you **must not** use the `puppet module` command to install or manage modules. Instead, you must declare modules in each environment's Puppetfile. Code management uses the Puppetfile to install, update, and manage your modules. If you use `puppet module install` to install a module to the live code directory, code management deletes the module when it is not found in the Puppetfile.

Declaring your own modules

If you develop your own modules that you maintain in source control, you can declare them in your Puppetfile, just like you would declare any module from a Git repository. If your modules aren't maintained in source control, you'll need to move them to source control so you can declare them in your Puppetfile and allow code management to install and manage your module in your environments.

Related information

- [Declare Git repositories in the Puppetfile](#) on page 771

Deploying module code

When you change your Puppetfile to install or update a module (or when you update a module that you wrote that you've declared in your Puppetfile), you must trigger Code Manager or r10k to deploy the new or updated code to your environments.

Tip: Code Manager does not automatically deploy modules' `spec` directories. These directories are for testing only, and they are not useful in a production environment. If you want to deploy a module's `spec` directory, add `exclude_spec: false` to the module declaration in your Puppetfile.

Related information

- [Triggering Code Manager on the command line](#) on page 795
- [Triggering Code Manager with a webhook](#) on page 801
- [Deploying environments with r10k](#) on page 831

Create a Puppetfile

The Puppetfile manages an environment's content. When you create a Puppetfile, use the `mod` directive to declare an environment's content.

Before you begin

You must be [Managing environments with a control repository](#) on page 763. These steps assume you have set up a control repository that has the production branch as the default branch.

These steps explain how to create an initial Puppetfile in your production environment (which is usually the default environment). This initial Puppetfile becomes a template for your other environments. When you [Add an environment](#) on page 767 (by creating a branch based on the default branch), the new environment inherits a copy of the default environment's Puppetfile, which you can then modify on the new branch to declare the new environment's content.

1. On your production branch, in the root directory, create a text file named `Puppetfile`.
2. Open the new `Puppetfile` in a text editor, such as VS Code.
3. Declare the production environment's content in the `Puppetfile`.

Use a `mod` directive to specify each module or repository. Additionally, you need to define the name of the content and any other information code management needs to correctly install and maintain the declared modules and data. For information and examples of Puppetfile declarations, refer to:

- [Declare Forge modules in the Puppetfile](#) on page 770
- [Declare Git repositories in the Puppetfile](#) on page 771

Tip: Puppet has a [VS Code extension](#) that supports syntax highlighting for the Puppet language.

4. Optional: If you want code management to install modules somewhere other than the default directory (`./modules`), use the `moduledir` directive to [Change the module installation directory](#) on page 774.
5. Save and commit your changes.

If you already have multiple branches (environments) in your control repo, you might need to copy the Puppetfile to the other branches, and then edit each copy according to each environment's module and data requirements. When you [Add an environment](#) on page 767, the new branch automatically gets a copy of the Puppetfile that you can then edit accordingly for the new environment.

Creating a Puppetfile is a requirement for [Managing code with Code Manager](#) on page 774 or [Managing code with r10k](#) on page 821.

Declare Forge modules in the Puppetfile

When you declare a Forge module in your Puppetfile, you can specify a particular version to track and whether you want code management to automatically update the module.

Important:

The Puppetfile **does not** automatically resolve dependencies for Forge modules. When you declare a module in your Puppetfile, you must also declare any required dependent modules.

Forge module symlinks **are not** supported. When you install modules with r10k or Code Manager, by declaring them in your Puppetfile, symlinks are not installed.

If you have Puppetfiles you used before you started using code management, these files might contain a `forge` setting that provides legacy compatibility with `librarian-puppet`. However, this setting is non-operational for Code Manager and r10k. If you need to configure how Forge modules are downloaded, you must specify `forge_settings` in Hiera. For instructions, refer to [Configuring Forge settings](#) on page 787 for Code Manager or [Configuring Forge settings](#) on page 825 for r10k.

1. In your Puppetfile, use the `mod` directive to specify Forge modules you want to install. Specify the module's full name as a string. For example, this declaration is for the `apache` module:

```
mod 'puppetlabs/apache'
```

Tip: This basic declaration installs the current version of the module that is available during the next code deployment, but it doesn't update the module on future runs. If you want to keep the module updated automatically, you need to specify `:latest`, as described in the next step.

2. Optional: Specify whether you want to maintain a specific version of the module or if you want code management to automatically update the module when a new version is available.

- To continuously keep the module current with the newest version, specify `:latest` after the module name. For example:

```
mod 'puppetlabs/ntp', :latest
```

- To install a specific version, and maintain that version, specify the desired version number, as a string, after the module name. For example:

```
mod 'puppetlabs/stdlib', '0.10.0'
```

- To install whichever version is current during the next code deployment, and stay with that version, do not specify any options after the module name. For example:

```
mod 'puppetlabs/apache'
```

3. Save and commit your changes.

Edit the Puppetfile any time you need to install a module or update a module that is not automatically updated.

With code management, you **must not** use the `puppet module` command to install or manage modules. Because code management uses the Puppetfile to install, update, and manage your modules, if you use `puppet module install` to install a module to the live code directory, code management deletes the module based on the Puppetfile contents.

Related information

[Managing modules with a Puppetfile](#) on page 768

Almost all Puppet manifests are kept in *modules*, which are collections of Puppet code and data that have a specific directory structure. With Puppet Enterprise (PE) code management, you only use the Puppetfile to install and manage modules.

Declare Git repositories in the Puppetfile

You can declare your own modules, modules that aren't from the Forge, data, or other non-module content that you want to install from Git repositories.

- To specify environment content from a Git repository, use the mod directive and specify the content name as a string. Then and use :git to specify the repository location, and :branch to reference a branch. For example:

```
mod 'apache',
  :git => 'https://github.com/puppetlabs/puppetlabs-apache'
  :branch => '<BRANCH_NAME>'
```

- Optional: Specify additional options or alternative configurations, if needed:

- For non-module content, you must [Specify installation paths for repositories](#) on page 771.

Important: Content is installed in the `modules` directory and treated as a module, unless you use the `:install_path` option. You **must** use `:install_path` for non-module content to keep your data separate from your modules.

- If the content requires SSH authentication, read about how to [Declare module or data content with SSH private key authentication](#) on page 772.
- By default, content from Git repositories stays updated with the repository's main branch, but you can [Keep repository content at a specific version](#) on page 772 and [Declare content from a relative control repo branch](#) on page 773.

- Save and commit your changes.

Related information

[Managing modules with a Puppetfile](#) on page 768

Almost all Puppet manifests are kept in *modules*, which are collections of Puppet code and data that have a specific directory structure. With Puppet Enterprise (PE) code management, you only use the Puppetfile to install and manage modules.

Specify installation paths for repositories

You can set individual installation paths for any Git repositories you declare in a 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 (which is either the default `modules` directory or a custom path if you [Change the module installation directory](#) on page 774).

In your Puppetfile, under the Git repository's mod directive, use the `:install_path` option to declare the location where you want to install the content. The path must a string and it must be relative to the Puppetfile's location. To install in the root directory, specify an empty value.

Content is installed into a subdirectory named after to the content's mod directive. For example, this declaration installs site data content from a Git repository into the `./hieradata` directory:

```
mod 'site_data',
  :git => 'git@git.example.com:site_data.git',
  :install_path => 'hieradata'
```

The final file path for this content is `./hieradata/site_data`.

As another example, this declaration installs site data content from a different Git repository into a `site_data` directory at the root:

```
mod 'site_data_2',
```

```
:git => 'git@git.example.com:site_data_2.git',
:install_path => ''
```

The final file path for this content is `./site_data`.

Declare module or data content with SSH private key authentication

To declare content protected by SSH private keys, declare the content as a Git repository, and then configure the private key setting in your code management tool.

1. Declare the Git repository in your Puppetfile, using the Git repo's SSH URL. For example:

```
mod 'myco/privatemod',
:git => 'git@git.example.com:myco/privatemod.git'
```

Note: If modifying the Puppetfile triggers a code deployment, expect the code deployment to fail. You must complete the next step to get a successful code deployment.

2. Configure the private key settings by modifying the following Code Manager or r10k parameters in Hiera:

- To set a key for all Git operations, use the private key setting under `git-settings`.
- To set a private key for an individual remote repository, set the private key in the `repositories` hash in `git-settings` for each specific remote.

For more information about these parameters, refer to [Configuring Git settings](#) on page 788 for Code Manager or [Configuring Git settings](#) on page 826 for r10k.

To make these changes, you must follow the steps described in [Customize Code Manager configuration in Hiera](#) on page 785 and [Customizing r10k configuration](#) on page 823.

After completing both steps, you might need to manually trigger a code deployment.

- [Triggering Code Manager on the command line](#) on page 795
- [Triggering Code Manager with a webhook](#) on page 801
- [Triggering Code Manager with custom scripts](#) on page 803
- [Deploying environments with r10k](#) on page 831

Keep repository content at a specific version

By default, content from Git repositories stays updated with the repository's main branch, but you can configure the Puppetfile to maintain repository content at a specific version.

To specify a particular repository version you want to track, use one of the following options in the Git repository's declaration in your Puppetfile. Setting one of these options maintains the repository at the specified version and deploys any updates made to that particular version.

- `ref`: Specifies the Git reference to check out. This option can reference either a tag, a commit, or a branch.
- `tag`: Specifies a certain tag associated with the repo. For example:

```
mod 'apache',
:git => 'https://github.com/puppetlabs/puppetlabs-apache',
:tag => '0.9.0'
```

- `commit`: Specifies a certain commit in the repo. For example:

```
mod 'apache',
:git => 'https://github.com/puppetlabs/puppetlabs-apache',
:commit => '8df51aa'
```

- `branch`: Specifies a certain branch of the Git repo or [Declare content from a relative control repo branch](#) on page 773. For example:

```
mod 'apache',
:git => 'https://github.com/puppetlabs/puppetlabs-apache',
```

```
:branch => 'proxy_match'
```

In addition to one of the above options, you can also [Set a default branch for content deployment](#) on page 773.

Declare content from a relative control repo branch

If you declare a Git repository to track a specific branch, you can also specify the :control_branch option, which allows you to deploy content from a control repo branch relative to the location of the Puppetfile.

Before you begin

The :control_branch option is a modification of the :branch option, which you can use to [Keep repository content at a specific version](#) on page 772.

Normally, :branch tracks a specifically-named repository branch, such as `testing`, or a specific `feature` branch. If you specify `:branch => :control_branch`, it locates and tracks a branch in the Git repository that has the same name as the control repo branch where the Puppetfile is located.

For example, if your Puppetfile is in the `production` branch, content from the Git repo's `production` branch is deployed. Similarly, if you copy this Puppetfile to your `testing` branch, the tracking from that branch follows the Git repo's `testing` branch.

Important: With :control_branch, when you create new branches, you don't have to edit the inherited Puppetfile as extensively, because the tracked branches remain relative. However, your Git repository branch names **must match** your control repo's branch names for the :control_branch option to work successfully. You might want to [Set a default branch for content deployment](#) on page 773 as a backup in case no matching branch is found.

Here is an example of a declaration using :control_branch:

```
mod 'hieradata',
  :git    => 'git@git.example.com:organization/hieradata.git',
  :branch => :control_branch
```

Set a default branch for content deployment

You can specify a default branch that code management can use if it can't deploy the specified ref, tag, commit, or branch.

Before you begin

You can't use :default_branch by itself. This option can only be used in conjunction with :ref, :tag, :commit, or :branch, which are used to [Keep repository content at a specific version](#) on page 772.

In the Puppetfile, in the content declaration, set the :default_branch option to the branch you want to deploy if your specified option fails. For example, this declaration tracks the :control_branch and uses the `main` branch as a backup if no matching branch is found.

```
mod 'hieradata',
  :git    => 'git@git.example.com:organization/hieradata.git',
  :branch => :control_branch,
  :default_branch => 'main'
```

Tip: Specifying a :default_branch is recommended when you [Declare content from a relative control repo branch](#) on page 773, in case code management can't find a matching branch.

If code management can't parse the default branch specification or no such named branch exists, it logs an error and does not deploy or update the content.

Change the module installation directory

If needed, you can change the directory to where code management installs modules declared in your Puppetfile.

By default, Code Manager and r10k install module content in a `modules` directory in the same directory the Puppetfile is in, such as: `./modules/<MODULE_NAME>`

To change the module installation path, at the top of your Puppetfile before any module declarations, add the `moduledir` directive, and specify the path to the desired module installation directory relative to the Puppetfile's location. For example:

```
moduledir 'thirdparty'
```

Important: This directive applies to **all** content declared in the Puppetfile.

If you need to change the installation paths for only some modules or data, declare those content sources as Git repositories, and use the `install_path` option to [Specify installation paths for repositories](#) on page 771. This option overrides the `moduledir` directive.

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 775

To automatically manage your environments and modules, Code Manager uses r10k and the file sync service to stage, commit, and sync your code.

- [Set up Code Manager](#) on page 778

You must set up Code Manager to use it as your code management tool.

- [Configure Code Manager](#) on page 778

To configure Code Manager you must enable Code Manager in Puppet Enterprise (PE), set up authentication, and test the connection between the control repository and Code Manager.

- [Configure Code Manager concurrency](#) on page 783

Enable Code Manager in Puppet Enterprise (PE), and then use the variables to configure Code Manager concurrency.

- [Lockless code deploys](#) on page 783

The *lockless code deploys* feature within Code Manager allows deployment of Puppet code without interrupting other Puppet operations. When this feature is disabled, requests to Puppet Server are blocked during code deployments until the file sync client has finished updating the live Puppet code directory. However, when lockless code deploys are enabled, the file sync client saves newly deployed code into versioned directories, ensuring that the live code directory is not overwritten. This process allows Puppet operations to continue without interruption during code deployments.

- [Customize Code Manager configuration in Hiera](#) on page 785

Set parameters in Hiera to customize your Code Manager configuration.

- [Triggering Code Manager on the command line](#) on page 795

Use the `puppet-code` command to trigger Code Manager from the command line and deploy your environments.

- [Triggering Code Manager with a webhook](#) on page 801

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. Webhooks are the simplest way to trigger Code Manager.

- [Triggering Code Manager with custom scripts](#) on page 803

Custom scripts are a good way to trigger deployments if you can't use webhooks. For example, if you have privately hosted Git repositories, custom notifications, or existing continuous integration systems (like Continuous Delivery for Puppet Enterprise (PE)).

- [Troubleshooting Code Manager](#) on page 804

Code Manager requires coordination between multiple components, including source control, r10k, and the file sync service. If you have issues with Code Manager, check that these components are functioning.

- [Code Manager API](#) on page 807

You can use the Code Manager API to deploy code and check the status of deployments on your primary server and compilers without direct shell access.

- [About file sync](#) on page 817

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

How Code Manager works

To automatically manage your environments and modules, Code Manager uses r10k and the file sync service to stage, commit, and sync your code.

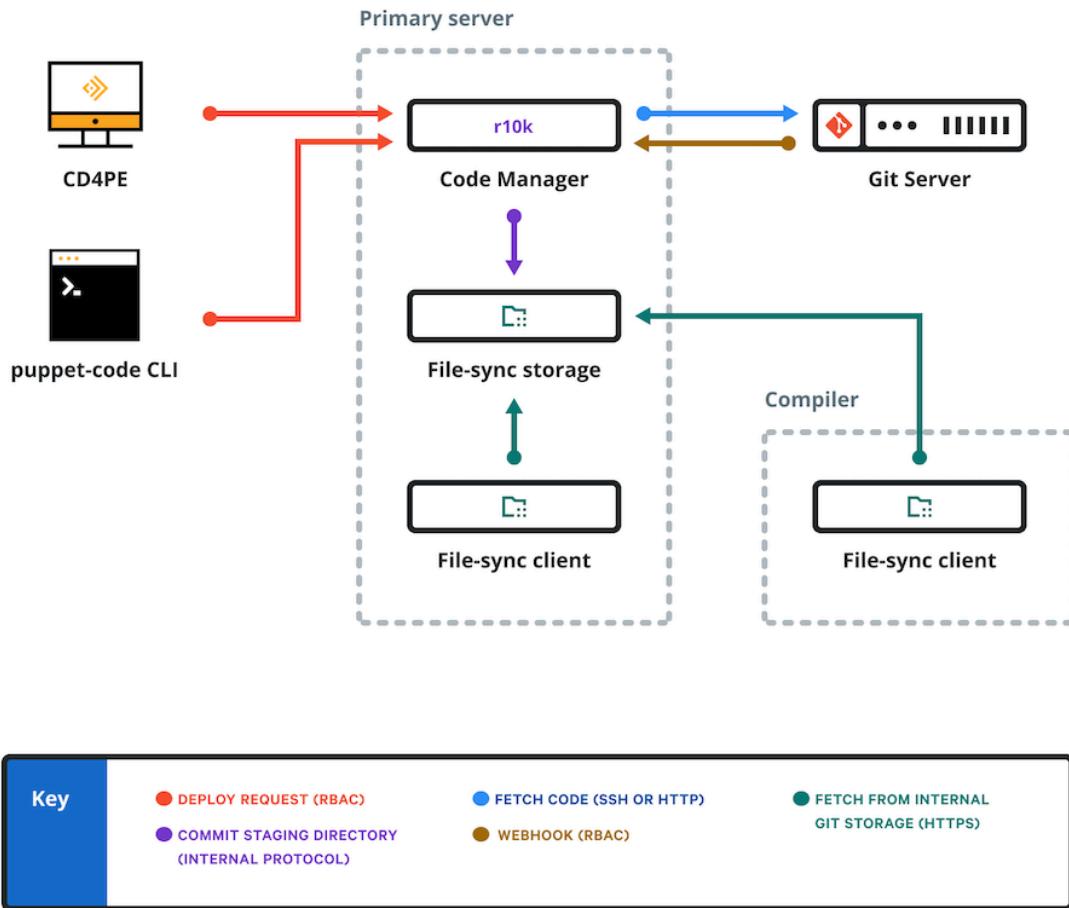
Code Manager requires [Managing environments with a control repository](#) on page 763. You must create a control repository with branches for each environment that you want to create (such as production, development, or testing). Each branch must have a Puppetfile specifying exactly which modules to install in each environment. You can learn more [About Environments](#) in the Puppet documentation.

Code Manager creates directory environments based on the branches you've set up. Your control repository lives on a Git server, and this is where you push code that you want Code Manager to deploy.

There are three ways to trigger Code Manager to start a code deployment:

- A webhook from your Git server automatically starts the code deployment when you push code to the control repo.
- [Continuous Delivery](#) sends a deploy request to Code Manager.
- You use the `puppet-code` command to manually trigger Code Manager from the command line or with a custom script.

Once triggered, Code Manager uses r10k to fetch code from the Git server and places it into the staging directory on the primary server (at `/etc/puppetlabs/code-staging`). Next, the file sync storage service on the primary server detects the change in the staging directory, and the file sync clients pause Puppet Server to avoid conflicts during synchronization. Finally, the file sync clients synchronize the new code to the live code directories on the primary server and compilers (usually at `/etc/puppetlabs/code`). The following diagram illustrates this code deployment process.



Related information

[About file sync](#) on page 817

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

[Triggering Code Manager on the command line](#) on page 795

Use the `puppet-code` command to trigger Code Manager from the command line and deploy your environments.

[Triggering Code Manager with a webhook](#) on page 801

To deploy your code, you can trigger Code Manager by hitting a web endpoint, either through a webhook or a custom script. Webhooks are the simplest way to trigger Code Manager.

[Triggering Code Manager with custom scripts](#) on page 803

Custom scripts are a good way to trigger deployments if you can't use webhooks. For example, if you have privately hosted Git repositories, custom notifications, or existing continuous integration systems (like Continuous Delivery for Puppet Enterprise (PE)).

Understanding file sync and the staging directory

To sync your code across your primary server and compilers, and to make sure that code stays consistent, Code Manager relies on file sync and two different code directories: the staging directory and the live code directory.

Without Code Manager or file sync, Puppet code lives in the `codedir`, or live code directory, at `/etc/puppetlabs/code`.

With Code Manager and file sync, the file sync client service regularly checks for changes to staged code files in the staged code directory on the primary server (at `/etc/puppetlabs/code-staging`). If it detects a change, the file sync client service fetches the changes and syncs the files to the live codedir on each compiler and the primary server (at `/etc/puppetlabs/code`). The file sync client service uses HTTPS to poll for changes and JGit to fetch changes.

Because Code Manager moves new code from source control into the staging directory, and file sync moves it into the live code directory, you no longer write code in the codedir. If you manually edit the codedir, the next time Code Manager deploys code from source control, it overwrites your changes.

Important: Don't directly modify code in the staging directory or live code directory (codedir). Code Manager overwrites the staging directory with changes from the control repo, and file sync overwrites the codedir with changes from the staging directory. Any changes made to these directories manually are overwritten.

Related information

[File sync terms](#) on page 817

Understanding these terms is helpful for understanding file sync.

Environment isolation metadata and Code Manager

The live code and staging code directories contain metadata files generated by file sync, which provide environment isolation for your resource types.

The metadata files, which have a `.pp` extension, ensure that each environment uses the correct version of the resource type.



CAUTION: Do not delete or modify the metadata files. Do not use expressions from these files in regular manifests.

These files are generated when Code Manager deploys new code in your environments. If you are new to Code Manager, these files are generated when you first deploy your environments. If you already use Code Manager, the files are generated as you make and deploy changes to your existing environments.

You can learn more about these files and their role in [Environment isolation](#) in the Puppet documentation.

Moving from r10k to Code Manager

Moving from r10k to Code Manager can improve automation of your code management and deployments.

While we recommend using Code Manager whenever possible, reasons you might not want to upgrade from r10k to Code Manager include:

- Code Manager **does not** allow you to manually deploy code with r10k. If you depend on the ability to deploy modules directly from r10k (with the `r10k deploy module` command), we recommend continuing to use r10k.
- Code Manager must deploy all control repositories to the same directory. If you use r10k to deploy control repositories to different directories, we recommend continuing to use r10k.
- Code Manager supports the shellgit provider, but only for HTTPS. It does not support system .SSH configuration or other shellgit options.
- Code Manager does not support post-deploy scripts.

If you rely on any of the above configurations Code Manager does not support, or if you are using a custom script to deploy code, carefully assess whether or not Code Manager can support your goals.

Related information

[Upgrade from r10k to Code Manager](#) on page 779

To upgrade from r10k to Code Manager, you must disable the previous r10k installation.

[Managing code with r10k](#) on page 821

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k

requires you to *manually* deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

Set up Code Manager

You must set up Code Manager to use it as your code management tool.

To set up Code Manager, you must:

1. Prepare for [Managing environments with a control repository](#) on page 763. This involves creating a Git control repository that has a Puppetfile.
2. Create a control repository with Git for your code.

Code Manager uses the control repo to maintain and deploy your Puppet code and data. You can also create separate deployment environments in your Puppet infrastructure by creating branches in your control repository (such as a development branch for a development environment). Code Manager tracks your environments and updates them according to the changes you make in your control repo.

The Puppetfile specifies which modules and data to install in your environment, including what versions to install, and where to download the modules or other content.

3. [Configure Code Manager](#) on page 778.
4. Optional: [Customize Code Manager configuration in Hiera](#) on page 785
5. Deploy environments with a deployment trigger (recommended) or from the command line. For the initial configuration, you might prefer to use the command line, and then set up an automated trigger.
 - [Triggering Code Manager on the command line](#) on page 795
 - [Triggering Code Manager with a webhook](#) on page 801
 - [Triggering Code Manager with custom scripts](#) on page 803

Related information

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

[Lockless code deploys](#) on page 783

The *lockless code deploys* feature within Code Manager allows deployment of Puppet code without interrupting other Puppet operations. When this feature is disabled, requests to Puppet Server are blocked during code deployments until the file sync client has finished updating the live Puppet code directory. However, when lockless code deploys are enabled, the file sync client saves newly deployed code into versioned directories, ensuring that the live code directory is not overwritten. This process allows Puppet operations to continue without interruption during code deployments.

Configure Code Manager

To configure Code Manager you must enable Code Manager in Puppet Enterprise (PE), set up authentication, and test the connection between the control repository and Code Manager.

To configure Code Manager:

1. Create a control repo with a Puppetfile, as explained in [Managing environments with a control repository](#) on page 763.
2. [Upgrade from r10k to Code Manager](#) on page 779, if applicable.
3. [Enable Code Manager](#) on page 779.
4. [Set up authentication for Code Manager](#) on page 76.
5. [Test the control repository](#) on page 781.
6. [Test Code Manager](#) on page 782.

Depending on your needs, you might need to configure additional [Code Manager settings](#) on page 782, enable [Lockless code deploys](#) on page 783, or [Customize Code Manager configuration in Hiera](#) on page 785.

Related information

[Set up Code Manager](#) on page 778

You must set up Code Manager to use it as your code management tool.

Upgrade from r10k to Code Manager

To upgrade from r10k to Code Manager, you must disable the previous r10k installation.

Code Manager cannot correctly install or update code if other tools run r10k.

1. Disable your previous r10k installation.
2. Disable any tools that automatically run r10k. Usually this is the `zack-r10k` module.

Note: When you upgrade to Code Manager, you can no longer manually use r10k or the `zack-r10k` module.

After disabling r10k, configure Code Manager.

Related information

[Moving from r10k to Code Manager](#) on page 777

Moving from r10k to Code Manager can improve automation of your code management and deployments.

Enable Code Manager

Set parameters in the console to enable Code Manager and connect your primary server to your Git repository.

Before you begin

Set up an SSH key to permit the `pe-puppet` user to access your Git repositories.

Important: If you are using Microsoft AzureDevOps (ADO), use HTTPS rather than SSH. ADO does not work using SSH.

The SSH key must be:

- Owned by the `pe-puppet` user.
- Located on the primary server.
- Located in a directory the `pe-puppet` user has permission to view, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`.

1. In the console, click **Node groups**, locate the **PE Master** node group, and set these parameters for the `puppet_enterprise::profile::master` class:
 - a) Set `code_manager_auto_configure` to `true` to enable Code Manager.
 - b) For `r10k_remote`, enter a string that is a valid SSH URL for your Git control repository, such as `git@<YOUR.GIT.SERVER.COM>:puppet/control.git`.

Important: Some Git providers have additional requirements for enabling SSH access. For example, BitBucket requires `ssh://` at the beginning of the SSH URL (such as `ssh://git@<YOUR.GIT.SERVER.COM>:puppet/control.git`). See your provider's documentation for this information.

- c) For `r10k_private_key`, enter a string specifying the path to the SSH private key that permits the `pe-puppet` user to access your Git repositories, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`.

Important: If your PE installation includes disaster recovery, you must also set the `puppet_enterprise::profile::master::r10k_private_key` parameter in `pe.conf`. This ensures that the `r10k` private key is synced to your primary server replica.

- d) If you want to enable lockless code deploys, ensure that the `versioned_deploys` parameter is set to `true`.

With the lockless code deploys feature enabled, code deployments are saved in versioned code directories, so that the live code directory is not overwritten. This process allows Puppet operations to continue during code deployments.

If you do not require lockless code deploys, set the value to `false`.

Tip: Enabling lockless code deploys will help to minimize disruptions associated with upgrading to future PE versions in which the feature will be enabled by default.

2. Click **Commit**.
3. Run Puppet on your primary server and all compilers.

Potential errors:

If you use **Run Puppet** in the console to trigger the Puppet run, the job, on the **Jobs** page, appears to fail due to underlying services being restarted. This error is not fatal and the **Reports** page shows the actual, successful result.

Additionally, if you run Puppet on your primary server and all compilers at the same time, the compilers' logs might report these errors:

```
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
```

These errors occur when Puppet Server is restarting when the compilers poll for new code, and they usually stop when Puppet Server finishes restarting on the primary server. You can ignore these errors while the primary server starts.

Set up authentication for Code Manager.

Set up authentication for Code Manager

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

Before requesting an authentication token, you must assign a user to the deployment role.

1. In the Puppet Enterprise (PE) console, create a deployment user.

Tip: Create a dedicated deployment user for Code Manager to use.

2. Add the deployment user to the **Code Deployers** role.

When you install PE, this role is automatically created with default permissions for code deployment and token lifetime management.

3. Click **Generate Password** to create a password for the deployment user.

Request an authentication token for deployments.

Related information

[Configure puppet-access](#) on page 303

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.

[Add a user to a user role](#) on page 279

When you add a user to a role, the user gains the permissions you assign to that role. A user can't do anything in PE until they have been assigned to at least one role. If users are assigned to multiple roles, they get all permissions from all roles they are assigned to.

[Assign a user group to a user role](#) on page 287

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

To securely deploy your code, request an authentication token for the deployment user.

The default lifetime for authentication tokens is one hour. You can use the `Override default expiry` permission set to change the token lifetime to a duration better suited for a long-running, automated process.

Use the `puppet-access` command to generate the authentication token.

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

Tip: You can specify additional settings in this command, such as the token file's location or your RBAC API URL, as explained in [Configuration file settings for puppet-access](#).

2. Enter the deployment user's username and password when prompted.

The generated token is stored in a file for later use. The default token storage location is `~/ .puppetlabs/token`. You can run `puppet-access show` to view the token.

Test the connection to the control repo.

Related information

[Set a token-specific lifetime](#) on page 308

If you want a token to have a different lifetime than the default lifetime, you can set a different lifetime when you generate the token. This allows you to keep one token for multiple sessions.

[Generate a token for use by a service](#) on page 307

If you need to generate a token that a Puppet Enterprise (PE) service can use, and the token doesn't need to be saved, use the `--print` option with the `puppet-access` command.

Test the control repository

To make sure Code Manager can connect to the control repository, test the connection to the repository.

From the command line, run: `puppet-code deploy --dry-run`

If the control repository is set up properly, this command fetches and displays a list of environments in the control repository as well as the total number of environments.

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.

From the command line, deploy one environment by running: `puppet-code deploy my_test_environment --wait`

If Code Manager is configured correctly, this command deploys the test environment and returns deployment results with the SHA (a checksum for the content stored) for the control repository commit.

If the deployment does not work, review the Code Manager configuration steps, or refer to [Troubleshooting](#) for help.

After fully enabling and configuring Code Manager, you can trigger Code Manager to deploy your environments.

You can:

- [Triggering Code Manager on the command line](#) on page 795
- [Triggering Code Manager with a webhook](#) on page 801
- [Triggering Code Manager with custom scripts](#) on page 803

Code Manager settings

After configuring Code Manager, you can adjust its settings in the **PE Master** node group in the `puppet_enterprise::profile::master` class.

Code Manager requires these options, unless otherwise noted:

`puppet_enterprise::master::file_sync::chown_code_to_pe_puppet`

By leaving this enabled users help ensure they do not hit a class of errors that can occur by committing Puppet code files with the wrong permissions (or at least have those errors resolved on the next Puppet run). However, some users have codedirs large enough and I/O throughput restrictive enough that they require disabling these executive resources in the compiler catalogs.

Default: `true`

Valid values: `true` or `false`

`puppet_enterprise::profile::master::code_manager_auto_configure`

Specifies whether to autoconfigure Code Manager and file sync.

Default: `false`

Setting this to `true` also sets `environment_timeout` to unlimited.

`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, and optional in all other cases.

Example: `"/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519"`

`puppet_enterprise::profile::master::r10k_proxy`

Optional proxy used by r10k when accessing the Forge. If empty, no proxy settings are used.

Restriction: If `r10k_proxy` is specified, you must use an HTTP URL for the `r10k_remote` parameter and all Puppetfile module entries.

Example: `"http://proxy.example.com:3128"`

More information: [Set proxies for Code Manager traffic](#) on page 229

Additional and alternative Code Manager proxy configurations: [Customize Code Manager configuration in Hiera on page 785](#) (specifically [Configuring proxies](#) on page 789 and [Configuring Forge settings](#) on page 787)

`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`

Setting for the *lockless code deploys* feature. Define the parameter to specify whether code is updated in versioned code directories instead of blocking requests and overwriting the live code directory.

Tip: Setting `versioned_deploys` to `true` will help to minimize disruptions associated with upgrading to future PE versions in which the lockless code deploys feature will be enabled by default in Code Manager.

Default: false

More information: [Lockless code deploys](#) on page 783

`puppet_enterprise::master::environment_timeout`

Specifies if and how long environments are cached, which can significantly reduce your Puppet Server's CPU usage. You can specify these values:

- No caching: 0
- Retain environment data caches indefinitely: `unlimited`
- Cache environments for a specified length of time after their last use: Any length of time, such as `5m`

Default when Code Manager is enabled: `5m`

Default when Code Manager is not enabled: 0

If `code_manager_auto_configure` is set to `true`: `unlimited`

More information: [Change the environment_timeout setting](#) on page 217

[Customize Code Manager configuration in Hiera](#) on page 785 explains how you can use Hiera to further customize your Code Manager configuration.

Configure Code Manager concurrency

Enable Code Manager in Puppet Enterprise (PE), and then use the variables to configure Code Manager concurrency.

Here are the variables you can use to configure Code Manager concurrency and a description of what they do.

Deploy pool

The deploy pool controls the number of workers that fetch environments off the queue. Each deploy pool worker thread shells out to r10k to deploy an environment.

Note: Each worker thread has a copy of the cache stored in: `/opt/puppetlabs/puppet/cache`.

The `puppet_enterprise::master::code_manager::deploy_pool_size` defaults to 2. Set this in Hiera.

Download pool

The download pool controls the number of r10k threads that check for new versions of a module in the environment.

The `puppet_enterprise::master::code_manager::download_pool_size` defaults to 4 . Set this in Hiera.

Lockless code deploys

The *lockless code deploys* feature within Code Manager allows deployment of Puppet code without interrupting other Puppet operations. When this feature is disabled, requests to Puppet Server are blocked during code deployments

until the file sync client has finished updating the live Puppet code directory. However, when lockless code deploys are enabled, the file sync client saves newly deployed code into versioned directories, ensuring that the live code directory is not overwritten. This process allows Puppet operations to continue without interruption during code deployments.

Important: Lockless code deploys require PE version 2021.2 or later.

With lockless deploys enabled, each new deploy writes code to *versioned directories* at:

```
/opt/puppetlabs/server/data/puppetserver/filesync/client/versioned-dirs/
puppet-code/
```

When the feature is enabled, the Puppet Server code directory is set up at `/etc/puppetlabs/puppetserver/code` and points, via symlink, to the most recent versioned code directory (at the versioned directories filepath specified above). If you disable lockless deploys, your code directory moves to `/etc/puppetlabs/code`.

Lockless deploys enable you to deploy a new version of code alongside an old version. When a catalog compiles starts, it uses the full path to the most recent version of code in the versioned code directory (via the `/etc/puppetlabs/puppetserver/code` symlink). Existing catalog compiles continue using the version they started on and new compiles use the latest code version.

To conserve disk space, code written to versioned directories is optimized to reduce duplication, and directories older than the latest and its predecessor are cleaned up after 30 minutes. If you deploy code very frequently, you might prefer to decrease the `versioned-dirs-ttl` setting, which is specified, in minutes, in `file-sync.conf` within each file sync client.

Related information

[Running plans alongside code deployments](#) on page 651

The orchestrator's file sync client has a built-in locking mechanism that ensures your plans run in a consistent environment state. The locking mechanism prevents plans from starting while a code deployment is in progress, and it prevents new code deployments from synchronizing while a plan is running. You can disable this locking mechanism if you want to run plans and deploy code simultaneously. Consider the tradeoffs before deciding whether to disable the file sync locking mechanism.

System requirements for lockless deploys

Enabling *lockless deploys* increases the disk storage required on your primary server and compilers because code is written to multiple versioned directories, instead of a single live code directory. Follow these guidelines for estimating your required system capacity.

You can roughly estimate your required disk storage with this equation:

$$(\text{Size of typical environment}) \times (\text{Number of active environments})$$

For example, if your typical environment is 200 MB on disk when deployed, and you have 25 active environments, your disk storage calculation is $200 \text{ MB} \times 25$, which equals 5,000 MB or 5 GB.

The number of times you deploy a given environment each day also impacts your disk use. Deploying multiple versions of the same environment uses approximately 25% more disk space than deploying multiple unique environments. To estimate the *additional* disk storage required for deploying environments multiple times a day, use this equation:

$$(\text{Size of typical environment} \times .25) \times (\text{Number of environments deployed multiple times per day}) \times (\text{Number of deployments per day})$$

Continuing the previous example, if 6 of your 200 MB environments are deployed up to 10 times per day, your additional disk storage calculation is $(200 \text{ MB} \times .25) \times 6 \times 10$, which equals 3,000 MB or 3 GB of additional disk space. In total, this example requires 8 GB available for your primary server and each compiler.

Note: If you're using the Continuous Delivery for PE impact analysis tool, you might need additional disk space beyond these estimates to accommodate the short-lived environments created during impact analysis.

Toggle lockless code deploys on or off

Use the `toggle_lockless_deploys` plan to turn lockless code deploys on or off across all compiler nodes, including the primary server and the replica.

Before you begin

For the toggle plan to run successfully, all compiler nodes (including the primary server and the replica) must have the same lockless code deploys status: the feature must be enabled on all or disabled on all. The plan cannot proceed if some nodes have the feature enabled while others have it disabled. To bring your compiler nodes into alignment, you can [Enable lockless code deploys on one compiler](#) on page 785.

You can use the `toggle_lockless_deploys` plan by running these commands on the primary server:

- To enable lockless code deploys:

```
puppet infra run toggle_lockless_deploys enable=true
```

- To disable lockless code deploys:

```
puppet infra run toggle_lockless_deploys enable=false
```

If you do not specify `true` or `false`, `true` is the default.

- Optionally, you can use the `old_code_directory` parameter with either the `save` or `delete` values, to specify whether you want to save old code directories to `#{codedir}_backup`. For example:

```
puppet infra run toggle_lockless_deploys enable=true
  old_code_directory=save
```

If you do not specify the `old_code_directory` parameter, `save` is set by default.

Enable lockless code deploys on one compiler

Use these steps to enable lockless code deploys on one Puppet Server instance.

You can follow this process for individual compiler nodes to ensure that all your compiler nodes have the same lockless code deploy status, which is a prerequisite for running the `toggle_lockless_deploys` plan. Or you might want to enable lockless code deploys on an individual compiler for testing, before you enable the feature across all compiler nodes.

- In the compiler's node-specific Hiera file, set `puppet_enterprise::profile::master::versioned_deploys` to `true` to enable the feature, or `false` to disable it.
- Commit changes.
- Run Puppet on the compiler twice: `puppet agent -t`; `puppet agent -t`
- On your primary server, run: `puppet code deploy --all --wait`

Important: You must deploy all environments (with `--all`) to avoid errors.

Tip: To monitor the impact on a compiler after enabling lockless code deploys, you can analyze the Puppet Server data collected by the [puppet_metrics_collector module](#).

Customize Code Manager configuration in Hiera

Set parameters in Hiera to customize your Code Manager configuration.

To customize your configuration:

- In your control repo, open the `data/common.yaml` file.

2. Add parameters to the `puppet_enterprise::master::code_manager` class. Use the following format:

```
puppet_enterprise::master::code_manager::<PARAMETER>: <SETTING>
```

For example, these parameters increase the size of the default worker pool and reduce the maximum time allowed to deploy a single environment:

```
puppet_enterprise::master::code_manager::deploy_pool_size: 4
puppet_enterprise::master::code_manager::timeouts_deploy: 300
```

Some parameters are described in detail below, along with a list of all [Code Manager parameters](#) on page 791.

3. Run Puppet on the primary server. The Puppet run updates the Code Manager configuration file.

Important: Do not manually edit the Code Manager configuration file. Puppet automatically manages this file, and it overwrites or discards any manual changes you make.

Related information

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Configuring post-environment hooks

Post-environment hooks can trigger custom actions after deploying an environment.

To configure list of hooks to run after Code Manager deploys code to an environment, specify the `post_environment_hook` parameter in Hiera. This parameter accepts an array of hashes with the `url` and `use-client-ssl` keys.

The `url` key specifies an HTTPS URL to send a POST request to. The request includes a JSON-formatted body containing information about the environment deployment, such as:

```
{
  "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
  "file-sync": {
    "environment-commit": "6939889b679fdb1449545c44f26aa06174d25c21",
    "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1" },
  "environment": "production",
  "id": 3,
  "status": "complete"
}
```

The `use-client-ssl` key is a Boolean specifying whether to use the client's SSL configuration for HTTPS connections.

By default, `use-client-ssl` is set to `false`, which means that when the HTTP client makes a request, it uses certificates from the Puppet Enterprise Java trust store file, which is located at:

```
/opt/puppetlabs/server/apps/java/lib/jvm/java/jre/lib/security/cacerts)
```

Important: When you upgrade PE, any certificates you added to this trust store file, are cleared. If the certificates are still required, you must add them again.

Set `use-client-ssl` to `true` only if the `url` destination is a server that uses the Puppet certificate authority.

For example, the following settings instruct Code Manager to update classes in the console after deploying code to environments:

```
puppet_enterprise::master::code_manager::post_environment_hooks:
```

```
- url: 'https://console.example.com:4433/classifier-api/v1/update-classes'
  use-client-ssl: true
```

If you wanted to configure multiple post-environment hooks, you would add more hashes to the array.

Related information

[POST /v1/update-classes](#) on page 558

Trigger the node classifier to retrieve updated class and environment definitions from the primary server. The classifier service also uses this endpoint when you refresh classes in the console.

[POST /v1/deploys](#) on page 808

Trigger Code Manager to deploy code to a specific environment or all environments, or use the `dry-run` parameter to test your control repo connection.

Configuring garbage collection

By default, Code Manager retains environment deployments in memory for one hour. You can adjust this by configuring garbage collection.

To configure the frequency of Code Manager garbage collection, specify the `deploy_ttl` parameter in Hiera. This parameter accepts a string that includes one of the following suffixes:

- `d`: Days
- `h`: Hours
- `m`: Minutes
- `s`: Seconds
- `ms`: Milliseconds

For example, `deploy_ttl: '30d'` configures Code Manager to keep deployments in memory for 30 days.

Similarly, `deploy_ttl: '48h'` retains deployments in memory for 48 hours.

The default setting is `1h` (one hour).

Important: If the value of `deploy_ttl` is less than the combined values of `timeouts_fetch`, `timeouts_sync`, and `timeouts_deploy`, then all completed deployments are retained indefinitely. This could significantly slow Code Manager's performance over time. Refer to [Code Manager parameters](#) on page 791 for information about the `timeouts_*` parameters.

Configuring module deployment scope

By default, Code Manager performs incremental deployments of module code. You can use the `full_deploy` parameter to change the module code deployment scope.

Incremental deploys only sync modules whose definitions (in the environment's Puppetfile) allow their version to "float" (such as Git branches) and modules whose definitions have been added or changed since the environment's last deployment. Incremental deploys do not support SVN modules.

If you want to deploy all module code regardless of change or float status, you can disable incremental deploys by setting the following parameter to `true`:

```
puppet_enterprise::master::code_manager::full_deploy
```

To re-enable incremental deploys, set the `full_deploy` parameter to `false`.

Configuring Forge settings

To configure how Code Manager downloads modules from the Forge, specify the `forge_settings` parameter in Hiera.

This parameter specifies where Forge modules are installed from, and it sets a proxy for all Forge interactions. The `forge_settings` parameter accepts a hash that can use the following keys:

- `baseurl`: Indicate where Forge modules are installed from. The default is `https://forgeapi.puppetlabs.com`.
- `authorization_token`: Specify the token for authenticating to a custom Forge server.
- `proxy`: Set the proxy for all Forge interactions.

For example, this configuration specifies a custom Forge server that doesn't require authentication:

```
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: 'https://private-forge.example'
```

If your custom Forge server requires authentication, you must specify both `baseurl` and `authorization_token`. You must format `authorization_token` as a string prepended with `Bearer`, particularly if you use Artifactory as your Forge server. For example:

```
puppet_enterprise::master::code_manager::forge_settings:
  baseurl: 'https://private-forge.example'
  authorization_token: 'Bearer <TOKEN>'
```

The `proxy` parameter sets a proxy for all Forge interactions. This setting overrides the global `proxy` setting but only for Forge operations (refer to the global `proxy` setting for more information). You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example:

```
puppet_enterprise::master::code_manager::forge_settings:
  proxy: 'http://proxy.example.com:3128'
```

Tip: If you set a global `proxy`, but you don't want Forge operations to use a proxy, under the `forge_settings` parameter, set `proxy` to an empty string.

Configuring Git settings

To configure Code Manager to use a private key, a proxy, or multiple Git source repositories, specify the `git_settings` parameter in Hiera.

Restriction: You can't use the `git_settings` parameter with the default Code Manager `r10k_private_key` settings. To avoid errors, remove the `r10k_private_key` parameter from the `puppet_enterprise::profile::master` class.

The `git_settings` parameter accepts a hash that can use the `private-key`, `proxy`, and `repositories` keys.

private-key

The `private-key` setting is required, and, if it is not specified, it gets a default value from the `puppet_enterprise::profile::master` class.

Use `private-key` to specify the path to the file containing the default private key that you want Code Manager to use to access control repos, for example:

```
/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

Important: The `pe-puppet` user must have read permissions for the private key file, and the SSH key can't require a password.

proxy

The `proxy` key sets a proxy specifically for Git operations that use an HTTP(S) transport. This setting overrides the global `proxy` setting but only for Git operations (For more information, refer to the global `proxy` setting). You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example:

```
proxy: 'http://proxy.example.com:3128'
```

Tip:

To set a proxy for only one specific Git repository (or when you have multiple control repos), set `proxy` within the `repositories` key.

If you set a global `proxy`, but you don't want Git operations to use a proxy, under the `git_settings` parameter, set `proxy` to an empty string.

repositories

The `repositories` key specifies a list of repositories and their respective private keys or proxies. Use `repositories` if:

- You need to configure different proxy settings for specific repos, instead of all Git operations.
- You have multiple control repos.

Important: If you have multiple control repos, the `sources` setting and the `repositories` setting must match.

The `repositories` setting accepts an array of hashes that use the `remote`, `private-key`, and `proxy` keys.

The `remote` key specifies the repository to which the subsequent `private-key` or `proxy` setting applies. The `private-key` and `proxy` settings have the same requirements and functions as described above, except that, when inside `repositories`, these settings only apply to a single repository.

For example, this `repositories` hash specifies unique private keys for two repos and a unique proxy for a third repo:

```
repositories:
  - remote: "jf kennedy@gitserver.puppetlabs.net:repositories/repo_one.git"
    private-key: "/test_keys/jfkennedy"
  - remote: "whtaft@gitserver.puppetlabs.net:repositories/repo_two.git"
    private-key: "/test_keys/whtaft"
  - remote: "https://git.example.com/git_repos/environments.git"
    proxy: "https://proxy.example.com:3128"
```

Tip: If you set a global proxy or a `git_settings` proxy, but you don't want a specific repo to use a proxy, in the `repositories` hash, set that specific repo's `proxy` to an empty string.

Configuring proxies

If you need Code Manager to use a proxy connection, use the `proxy` parameter. You can set a global proxy for all HTTP(S) operations, proxies for Git or Forge operations, or proxies for individual Git repositories.

Where you specify the `proxy` parameter depends on how you want to apply the setting:

- To set a proxy for all Code Manager operations occurring over an HTTP(S) transport, set the global `proxy` setting.
- To set proxies only for Git operations or individual Git repos, set the appropriate `proxy` key under the `git_settings` parameter.
- To set a proxy only for Forge operations, set the `proxy` key under the `forge_settings` parameter.

You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example, this setting is for an unauthenticated proxy:

```
proxy: 'http://proxy.example.com:3128'
```

Whereas this setting is for a password-authenticated proxy:

```
proxy: 'http://user:password@proxy.example.com:3128'
```

Override proxy settings

You can override the global proxy setting if you want to:

- Set a different proxy setting for Git or Forge operations.
- Specify a different proxy setting for an individual Git repo.
- Specify a mix of proxy and non-proxy connections.

To override the global proxy setting for all Git or Forge operations, you need to set the `proxy` key under the `git_settings` or `forge_settings` parameters.

To set a proxy for an individual Git repository (or if you have multiple control repos), set the `proxy` key in the `repositories` hash under the `git_settings` parameter.

If you have set a global, Git, or Forge proxy, but you **don't** want a certain setting to use any proxy, set the `proxy` parameter to an empty string. For example, if you set a global proxy, but you don't want Forge operations to use a proxy, you would specify an empty string under the `forge_settings` parameter, such as:

```
puppet_enterprise::master::code_manager::forge_settings:
  proxy: ''
```

Tip: You can use curl commands to test Git and Forge proxy connections, such as:

```
curl --proxy "<YOUR_PROXY_URI>" --head "https://github.com"
curl --proxy "<YOUR_PROXY_URI>" --head "https://forgeapi.puppet.com"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Related information

[Set proxies for Code Manager traffic](#) on page 229

Code Manager has proxy configuration options you can use to set proxies for connections to your Git server, the Forge, specific Git repositories, or all Code Manager operations over HTTP(S) transports.

Configuring sources

If you are managing multiple control repos with Code Manager, you must use the `sources` parameter to specify a map of your source repositories.

The `sources` parameter is necessary when Code Manager is managing multiple control repos. For example, your Puppet environments are in one control repo and your Hiera data is in a separate control repo.

Important:

The `sources` setting and the `repositories` setting (under `git_settings`) must match.

If `sources` is set, you can't use Code Manager's global `remote` parameter.

The `sources` parameter consists of a list of source names along with a hash containing the `remote` and `prefix` key for each source. For example:

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: "testing"
```

The `remote` parameter specifies the location from which to fetch the source repo. Code Manager must be able to fetch the remote without any interactive input. This means fetching the source can't require inputting a user name or password. You must supply a valid URL, as a string, that Code Manager can use to clone the repo, such as: `"git://git-server.site/myorg/main-modules"`

The `prefix` parameter specifies a string to use as a prefix for the names of environments derived from the specified source. Set this to a specific string if you want to use a specific prefix, such as `"testing"`. Set this to `true` to use the source's name as the prefix. The `prefix` parameter prevents collisions (and confusion) when multiple sources with identical branch names are deployed into the same directory.

For example, the following settings might cause errors or confusion because there would be two `main-modules` environments deployed to the same base directory:

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  prefix: true
mysource:
  remote: "git://git-server.site/mysource/main-modules"
  prefix: true
```

However, by changing one `prefix` to `"testing"`, the two environments become more distinct, since the directory would now have a `main-modules` environment and a `testing-main-modules` environment:

```
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

Parameter	Description	Type	Default value
<code>r10k_remote</code> (also referred to as Code Manager's global <code>remote</code>)	A valid SSH URL specifying the location of your Git control repository, if you have only one control repo. If you have multiple Git repos, specify <code>sources</code> instead of <code>remote</code> . If you specify both <code>sources</code> and <code>remote</code> , then <code>sources</code> overrides <code>remote</code> .	String	If <code>r10k_remote</code> is specified in the <code>puppet_enterprise::profile::...</code> class, that value is used here. Otherwise, there is no default value.

Parameter	Description	Type	Default value
authenticate_webhook	Indicates whether to enable RBAC authentication for the POST /v1/webhook on page 813 endpoint.	Boolean	true
cachedir	The file path to the location where Code Manager caches Git repositories.	String	/opt/puppetlabs/server/data/code-manager/cache
certname	The certname of the Puppet signed certs to use for SSL	String or string variable	\$::clientcert
data	The file path to the directory where Code Manager stores internal file content.	String	/opt/puppetlabs/server/data/code-manager
deploy_pool_cleanup	Specifies how often workers pause to clean their on-disk caches. If cleanup takes too long, increase this value so that cleanup happens less often.	Integer indicating a 1 out of n percent chance.	100 (Cleanup occurs after 1 of every 100 code deployments, or after 1% of code deployments.)
deploy_pool_size	Specifies the number of threads in the worker pool, which determines how many deployment processes can run in parallel.	Integer	2
download_pool_size	Specifies the number of threads used to download modules.	Integer	4
deploy_ttl	For Configuring garbage collection on page 787.	String with a required suffix	1h
full_deploy	For Configuring module deployment scope on page 787.	Boolean	false
hostcrl	The file path to the SSL CRL.	String or string variable	\$puppet_enterprise::params::hostcrl
localcacert	The file path to the SSL CA cert.	String or string variable	\$puppet_enterprise::params::localcacert
post_environment_hooks	For Configuring post-environment hooks on page 786, which are hooks that you want to run after Code Manager deploys an environment.	Array of hashes	No default.
timeouts_deploy	Maximum execution time (in seconds) allowed for deploying a single environment.	Integer	600

Parameter	Description	Type	Default value
timeouts_fetch	Maximum execution time (in seconds) allowed for updating the control repo state.	Integer	30
timeouts_hook	Maximum time (in seconds) to wait for a single post-environment hook URL to respond. Controls both the socket connect timeout and the read timeout; therefore, the longest total timeout is twice the specified value.	Integer	30
timeouts_shutdown	Maximum time (in seconds) to wait for in-progress deployments to complete when shutting down the service.	Integer	610
timeouts_wait	Maximum time (in seconds) to wait for the environment's deployment to finish before timing out. Only applies to requests sent with the <code>wait</code> key.	Integer	700
timeouts_sync	Maximum time (in seconds) to wait for all compilers to receive deployed code before timing out. Only applies to requests sent with the <code>wait</code> key.	Integer	300
webserver_ssl_host	The IP address of the host that Code Manager listens on.	IP address	0.0.0.0
webserver_ssl_port	The port that Code Manager listens on.	Integer	8170
Important: Port 8170 must be open if you're using Code Manager.			

r10k-specific parameters

Code Manager uses r10k in the background. In the context of Code Manager, the following r10k parameters apply.

Parameter	Description	Type	Default value
environmentdir	The file path to the single directory where Code Manager deploys all sources.	String	If <code>file_sync_auto_commit</code> is set to <code>true</code> , then this defaults to: <code>/etc/puppetlabs/code-staging/environments</code>
forge_settings	For Configuring Forge settings on page 787.	Hash	No default.
invalid_branches	Specifies how you want Code Manager to handle branch names that can't cleanly map to Puppet environment names.	Either of these strings: <ul style="list-style-type: none">• <code>'error'</code>: Ignore branches that have non-word characters, and report an error about the invalid branches• <code>'correct'</code>: Without providing a warning, replace non-word characters with underscores	<code>'error'</code>
git_settings	For Configuring Git settings on page 788.	Hash	Can use the default <code>private-key</code> value set in console. Otherwise, there are no default settings.
proxy	For Configuring proxies on page 789. Can be global (all HTTP(s) transports) or part of the <code>git_settings</code> or <code>forge_settings</code> hashes.	An empty string or a string indicating a proxy server (with or without authentication)	No default.
sources	For Configuring sources on page 790 when you have multiple control repos.	Hash	No default.

Triggering Code Manager on the command line

Use the `puppet-code` command to trigger Code Manager from the command line and deploy your environments.

Installing and configuring `puppet-code`

Puppet Enterprise (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 workstation, customize configuration for different users, or change the global configuration settings.

The global configuration settings for *nix and macOS systems are in a JSON file located at:

```
/etc/puppetlabs/client-tools/puppet-code.conf
```

By default, this configuration file contains:

```
{
  "cacert": "/etc/puppetlabs/puppet/ssl/certs/ca.pem",
  "token-file": "~/.puppetlabs/token",
  "service-url": "https://<PRIMARY_HOSTNAME>:8170/code-manager"
}
```

On Windows systems, the global configuration settings are located at: C:\ProgramData\PuppetLabs\client-tools\puppet-code.conf

On Windows, the default configuration file contains:

```
{
  "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"
}
```

Important: On PE-managed machines, Puppet manages this file for you. Don't manually edit this file, because Puppet overwrites your changes the next time it runs.

In addition to the global settings, you can:

- [Configure puppet-code on agents and workstations](#) on page 797
- [Configure puppet-code for different users](#) on page 797
- Use the command line to override specific configuration settings for one deployment

When [Deploying environments with puppet-code](#) on page 796, you can use the default config file, an alternative config file, or config settings supplied directly in the command.

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 [Use a temporary puppet-code.conf file](#) on page 796, Puppet temporarily uses that configuration file **only**. In this case, Puppet doesn't read the global or user-specific configuration files **at all** for that one deployment.

If you [Use a temporary cacert, token-file, or service-url](#) on page 796, by specifying individual configuration options directly on the command line, those options temporarily take precedence over any place they are specified in default, global, or user-specific configuration file settings. Settings you don't specify in this way are applied according to their normal configuration precedence.

Deploying environments with `puppet-code`

Use `puppet-code deploy` to trigger a Code Manager code deployment.

You must supply a specific environment's name or the `--all` flag (to deploy all environments). For example:

```
puppet-code deploy production
puppet-code deploy --all
```

Without any other options specified, the default `puppet-code deploy <ENVIRONMENT_OPTION>` command deploys the specified environment(s) and returns only *deployment queuing* results.

In addition to the options for `--wait` and custom configuration settings (described below), use the [Reference: puppet-code command](#) on page 798 to learn about other `puppet-code` options.

Running `puppet-code` on Windows

When running `puppet-code` on 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 code deploy production --wait
```

For more information about Windows modifications, refer to [Using example commands](#) on page 28.

Return deployment results (`--wait`)

If you want `puppet-code deploy` to return the results of the actual deployment event(s), add the `--wait` flag. Otherwise, the command returns only deployment queuing information.

For example:

```
puppet-code deploy --all --wait
```

With the `--wait` flag, Code Manager deploys code to the specified environment(s), and only returns results after file sync has deployed code to the live code directory and all compilers.

The resulting message includes the deployment signature, which is the commit SHA from 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.

Note: In deployments that are geographically dispersed or have a large quantity of environments, complete code deployment might take several minutes.

Use a temporary `puppet-code.conf` file

You can use a custom configuration file to temporarily override default, global, and user-specific configuration settings by specifying the temporary file on the command line.

If you want to temporarily override default, global, and user-specific configuration settings, use the `--config-file` option to specify the file path to an alternative `puppet-code.conf` file. For example:

```
puppet-code --config-file ~/.puppetlabs/myconfigfile/puppet code.conf deploy
--all
```

This configuration file is only used for this one deployment.

Use a temporary `cacert`, `token-file`, or `service-url`

You can temporarily override individual `puppet-code` configuration settings by specifying individual settings on the command line.

If you want to temporarily override default, global, and user-specific configuration settings, you can specify these settings directly on the command line:

- `--cacert`
- `--token-file` or `-t`

- `--service-url`

For example, this command uses a custom URL to call the Code Manager service:

```
puppet-code --service-url "https://puppet.example.com:8170/code-manager"
deploy production
```

Tip: For Windows, macOS, and *nix argument formatting examples, refer to [puppet-code configuration settings](#) on page 800.

When you specify settings this way, your custom settings are only used for this one deployment. Unspecified settings aren't overridden.

Advanced puppet-code configuration

You can configure the `puppet-code` command on agent nodes, workstations not managed by PE, and for individual users (on any machine).

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.

1. On the agent node or workstation, 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`
2. If this machine is not managed by PE, edit the `puppet-code.conf` file as needed to customize the `cacert`, `token-file`, and `service-url` settings. These must use proper JSON formatting.

Important: On PE-managed machines, Puppet manages this file for you. Don't manually edit this file on PE-managed machines, because Puppet overwrites your changes the next time it runs. However, you can apply temporary modifications, or use an alternative config file, when [Deploying environments with puppet-code](#) on page 796.

Related information

[Installing client tools](#) on page 172

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.

[Installing and configuring puppet-code](#) on page 795

Puppet Enterprise (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 workstation, customize configuration for different users, or change the global configuration settings.

Configure puppet-code for different users

On any machine, you can configure `puppet-code` settings for individual users.

Before you begin

If PE is **not** installed on the workstation you are configuring, you must [Configure puppet-code on the workstation](#) first.

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/` directory.
- For Windows systems, place the file in the default user config file location at: `C:\Users\<username>\.puppetlabs\ssl\certs\ca.pem`

- In the user's `puppet-code.conf` file, customize the `cacert`, `token-file`, and `service-url` settings as needed. These must use proper JSON formatting.

Related information

[Installing and configuring puppet-code](#) on page 795

Puppet Enterprise (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 workstation, customize configuration for different users, or change the global configuration settings.

[Configuration precedence and puppet-code](#) on page 795

There are several ways to configure `puppet-code`, but some configuration methods take precedence over others.

Reference: `puppet-code` command

The `puppet-code` command accepts options, actions, and `deploy` action options.

Use the following format to modify the `puppet-code` command:

```
puppet-code [GLOBAL_OPTIONS] <ACTION> [ACTION_OPTIONS]
```

Global `puppet-code` options

The `puppet-code` command supports these global options.

Option	Description	Allowed arguments
<code>--help</code> or <code>-h</code>	Prints <code>puppet-code</code> usage information.	No arguments supported
<code>--version</code> or <code>-V</code>	Prints the application's version.	No arguments supported
<code>--log-level</code> or <code>-l</code>	Sets the log verbosity.	One of the following log levels: <ul style="list-style-type: none">• <code>none</code>• <code>trace</code>• <code>debug</code>• <code>info</code>• <code>warn</code>• <code>error</code>• <code>fatal</code>
<code>--config-file</code> or <code>-c</code>	Specifies a <code>puppet-code.conf</code> file that takes precedence over all other existing <code>puppet-code.conf</code> files. Refer to: Use a temporary puppet-code.conf file on page 796	A path to a <code>puppet-code.conf</code> file

Option	Description	Allowed arguments
--cacert	Specifies a Puppet CA certificate that overrides the cacert setting in any configuration files. Refer to: Use a temporary cacert, token-file, or service-url on page 796	A path to the location of the CA Certificate
--token-file or -t	Specifies an authentication token that overrides the token-file setting in any configuration files. Refer to: Use a temporary cacert, token-file, or service-url on page 796	A path to the location of the authentication token
--service-url	Specifies a base URL for the Code Manager service, overriding the service-url setting in any configuration files. Refer to: Use a temporary cacert, token-file, or service-url on page 796	A valid URL to call the Code Manager service

puppet-code actions

The puppet-code command can perform these actions.

Action	Description	Action options
deploy	Triggers the Code Manager service to deploy code. Refer to: Deploying environments with puppet-code on page 796	Refer to: puppet-code deploy action options on page 800
print-config	Prints the resolved puppet-code configuration.	No action options supported

Action	Description	Action options
status	Checks whether Code Manager and file sync are responding.	<p>You can specify a log level:</p> <ul style="list-style-type: none"> • none • trace • info • warn • error • fatal <p>If unspecified, the default is info.</p>

puppet-code deploy action options

You can use these action options to modify the `puppet-code deploy` action.

Option	Description
--all or an environment name	Required. You must specify either a single environment's name or use <code>--all</code> to deploy all environments.
--dry-run	Tests the connections to each configured remote and, if successfully connected, returns a consolidated list of the environments from all remotes. The <code>--dry-run</code> flag implies both <code>--all</code> and <code>--wait</code> .
--format or -F	Applies pretty printing to the response.
--wait or -w	Refer to: Return deployment results (--wait) on page 796

puppet-code configuration settings

You can temporarily override `puppet-code.conf` settings on the command line.

Setting	Description	*nix and macOS default value	Windows default value
cacert	Specifies the path to the Puppet CA certificate to use when connecting to the Code Manager service over SSL.	/etc/puppetlabs/puppet/ssl/certs/ca.pem	C:\ProgramData\PuppetLabs\puppet\etc\ssl\certs\ca.pem
token-file	Specifies the location of the file containing the authentication token for Code Manager.	~/.puppetlabs/token	C:\Users\<USERNAME>\.puppetlabs\token
service-url	Specifies the base URL to call the Code Manager service.	https://<PRIMARY_HOSTNAME>:8170/code-manager	https://<PRIMARY_HOSTNAME>:8170/code-manager

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. Webhooks are the simplest way to trigger Code Manager.

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.

Tip: [Triggering Code Manager with custom scripts](#) on page 803 is a good alternative to webhooks if you have requirements such as existing continuous integration systems (including Continuous Delivery for Puppet Enterprise (PE)), privately hosted Git repos, or custom notifications.

Create 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.

Create a custom URL

Create a custom URL to allow 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).

Important: If you want to use a GitHub webhook with the Puppet signed cert, you must disable SSL verification.

The custom webhook URL must use these elements:

Element	Description	Example
Name	Your primary server's DNS name	puppet.example.com
Port	The port used by Code Manager	8170
Endpoint	The Code Manager webhook endpoint	/code-manager/v1/webhook/
Parameters	Required and optional Code Manager webhook query parameters on page 802	type=github&token=<TOKEN>

The minimum possible valid webhook URL format is:

```
https://<NAME>:<PORT>/code-manager/v1/webhook?type=<TYPE>
```

The `token` parameter is required *unless* you disabled `authenticate_webhook`. With the `token` parameter, the valid URL format is:

```
https://<NAME>:<PORT>/code-manager/v1/webhook?type=<TYPE>&token=<TOKEN>
```

If your source configuration requires the `prefix` parameter, the valid URL format is:

```
https://<NAME>:<PORT>/code-manager/v1/webhook?
type=<TYPE>&prefix=<PREFIX>&token=<TOKEN>
```

For example, the following URLs are for a GitHub webhook and a Bitbucket Server webhook:

```
https://puppet.example.com:8170/code-manager/v1/webhook?
type=github&token=<TOKEN>
```

```
https://puppet.example.com:8170/code-manager/v1/webhook?type=bitbucket-server&prefix=dev&token=<TOKEN>
```

Tip: After you [Set up the Code Manager webhook on your Git host](#) on page 803, you can use this URL to call the [POST /v1/webhook](#) on page 813 endpoint.

Code Manager webhook query parameters

You can use these query parameters in your the Code Manager webhook URL.

Parameter	Description	Value
type	Required. Specifies the type of POST body to expect.	Specify the value corresponding with your Git host: <ul style="list-style-type: none"> GitHub: type=github GitLab: type=gitlab Bitbucket Server version 5.4 or later (formerly Stash): type=bitbucket-server Bitbucket: type=bitbucket Team Foundation Server (resource version 1.0 is supported): type=tfs-git
prefix	Conditionally required. Specifies a prefix for converting branch names to environment names. Important: Required if you used prefixing when Configuring sources on page 790. If your sources use prefixing and you do not specify this parameter, Code Manager can't correctly locate or deploy environments, or translate branch names to valid environment names.	prefix=<PREFIX>
token	Conditionally required. Specifies the entire PE authorization token to use for code deployments. To get a token, you can Request an authentication token for deployments on page 77. Important: Required <i>unless</i> you disabled <code>authenticate_webhook</code> in your Code Manager configuration.	token=<TOKEN>

Related information

[Configuring sources](#) on page 790

If you are managing multiple control repos with Code Manager, you must use the `sources` parameter to specify a map of your source repositories.

Set up the Code Manager webhook on your Git host

In your Git server's webhook form, enter your custom URL as the payload URL.

The content type for Code Manager webhooks is JSON.

The specific steps for setting up a webhook depends on your Git host. Refer to your Git host's documentation for instructions.

For example, in a GitHub repo, click **Settings > Webhooks & services**, 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**. Use the **Webhooks** configuration for your Code Manager webhook. Make sure you're using Bitbucket Server version 5.4 or later and the latest fix version of PE.

After setting up your webhook, you've finished setting up Code Manager. From now on, when you commit new code and push it to your control repo, the webhook triggers Code Manager to deploy your code. You can also use the [POST /v1/webhook](#) on page 813 endpoint to manually trigger your webhook.

Troubleshoot a Code Manager webhook

To troubleshoot your webhook, you can review your Git host's logs. Refer to your Git host's documentation for information about their logs and their suggestions for resolving common webhook issues.

Deployments triggered by a webhook in Stash/Bitbucket, GitLab, or GitHub are governed by authentication and hit the [POST /v1/webhook](#) on page 813 endpoint for each service type.

If you are using a GitLab version older than 8.5.0, Code Manager's webhook authentication doesn't work because of the length of the authentication token. To use the webhook with GitLab, either disable authentication or update GitLab. If you disable webhook authentication, it is disabled **only** for the [POST /v1/webhook](#) on page 813 endpoint. It is not possible to disable authentication for the [POST /v1/deploy](#)s on page 808 or [GET /v1/deploy](#)s/[status](#) on page 814 endpoints.

To troubleshoot webhook issues, follow the instructions for [Triggering Code Manager with a webhook](#) on page 801 while monitoring the Puppet Server log. To monitor the logs, open a separate 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 webhook. The `puppetserver.log` file is the only location these errors appear. If you cannot determine the problem with your webhook this way, manually deploy to the [POST /v1/deploy](#)s on page 808 endpoint while monitoring the `console-services.log` file, as described in [Troubleshooting Code Manager](#) on page 804.

For other Code Manager issues, refer to [Troubleshooting Code Manager](#) on page 804.

Triggering Code Manager with custom scripts

Custom scripts are a good way to trigger deployments if you can't use webhooks. For example, if you have privately hosted Git repositories, custom notifications, or existing continuous integration systems (like Continuous Delivery for Puppet Enterprise (PE)).

[Triggering Code Manager with a webhook](#) on page 801 is simpler than using a custom script. If you can use a webhook, we recommend it.

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 Code Manager API endpoints. We recommend using the `puppet-code` command, if possible.

After pushing new code to your control repo, run your script to trigger Code Manager to deploy your code.

The following instructions assume you've [Set up Code Manager](#) on page 778.

Related information

[Reference: puppet-code command](#) on page 798

The `puppet-code` command accepts options, actions, and `deploy` action options.

[Code Manager API](#) on page 807

You can use the Code Manager API to deploy code and check the status of deployments on your primary server and compilers without direct shell access.

[Request an authentication token for deployments](#) on page 77

To securely deploy your code, request an authentication token for the deployment user.

Scripting the `puppet-code` command

The `puppet-code` command can deploy environments from the command line. You can use this command in custom Code Manager scripts.

Using `puppet-code` in Code Manager scripts is much the same as [Triggering Code Manager on the command line](#) on page 795. The benefit to using a script is that you don't need to rebuild the commands from scratch each time, since they are stored in scripts.

Build your desired `puppet-code` command with the relevant action, environment (or `--all` for all environments), custom `puppet-code.conf` settings, and other options. Then, incorporate the command into your complete script.

After pushing new code to your control repo, run your script to trigger Code Manager to deploy your code.

Related information

[Reference: puppet-code command](#) on page 798

The `puppet-code` command accepts options, actions, and `deploy` action options.

Scripting `deploys curl` commands

The Code Manager API `deploys` endpoint can trigger code deployments. You can use a curl command in your custom scripts to hit this endpoint.

Calling the `deploys` endpoint in a script is similar to forming a one-time call to the endpoint. By storing commands in scripts, you don't need to rebuild the commands from scratch each time.

Build your desired curl command with the relevant URI path, parameters, and authentication, as described in [POST /v1/deploys](#) on page 808. Then, incorporate the command into your complete script.

Scripting `deploys/status curl` commands

The Code Manager API `deploys/status` endpoint can check a deployment's status. You can use a curl command in your custom scripts to hit this endpoint.

Calling the `deploys/status` endpoint in a script is similar to forming a one-time call to the endpoint. By storing commands in scripts, you don't need to rebuild the commands from scratch each time.

Build your desired curl command with the relevant URI path, parameters, and authentication, as described in [GET /v1/deploys/status](#) on page 814. Then, incorporate the command into your complete script.

Troubleshooting Code Manager

Code Manager requires coordination between multiple components, including source control, 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 at: `/var/log/puppetlabs/puppetserver/puppetserver.log`

For more information about working with the logs, see the [Puppet Server logs](#) documentation.

For source control webhook issues, check your Git host's logs.

Check Code Manager's 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 `puppet-code status` command verifies that Code Manager and file sync are responding. The command returns the same information as the [GET /v1/deployments/status](#) on page 814 endpoint. By default, the command returns details at the `info` log level. It doesn't support `critical` and `debug` log levels.

Errors that `puppet-code status` might report include:

Code Manager couldn't connect to the server

Occurs if the `pe-puppetserver` process isn't running.

Code Manager reports invalid configuration

Occurs if there is an invalid configuration in the `code-manager.conf` file, located at: `/etc/puppetlabs/puppetserver/conf.d/code-manager.conf`

File sync storage service reports unknown status

Occurs if the status callback timed out.

Test the connection to the control repository

The control repository controls the existence of environments, and ensures that the correct versions of all the necessary modules are installed in your environments. 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 found in the control repo (or repos, if you have multiple sources configured). A successful response means the control repo's SSH key has the correct permissions, the Git URL is correct, and the `pe-puppet` user can perform the necessary operations.

If there is a problem with the connection, the command returns this message: `Unable to determine current branches for Git source`. It also returns a file path on the primary server that you can use for debugging the SSH key and Git URL.

Check the Puppetfile for errors

Check the environment's 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 an environment's Puppetfile in a temporary directory.

On the primary server, create a temporary directory at `/var/tmp/test-puppetfile` and place a copy of the Puppetfile into the temporary directory. From there, you can then check the syntax and sources in your Puppetfile.

To check the Puppetfile syntax, run `r10k puppetfile check` from within the temporary directory. If syntax errors are detected, correct them, and run the test again. If the syntax is correct, the command returns `Syntax OK`.

To test the configuration of sources in your Puppetfile, perform a test installation. In your temporary directory (at `/var/tmp/test-puppetfile`), run the following command:

```
sudo -H -u pe-puppet bash -c \
'/opt/puppetlabs/puppet/bin/r10k puppetfile install'
```

This command attempts to install modules listed in your Puppetfile to a `modules` directory in your temporary directory. This test verifies if there is access to all listed module sources. Take note of **all** errors that occur, because issues with individual modules can cause issues for the entire environment. Errors with an individual module (such as Git URL syntax or version issues) are reported as general errors for that module. If you have modules from private

Git repositories that require an SSH key to clone the module, check that you are using the SSH Git URL and not the HTTPS Git URL. After correcting the reported errors, rerun the test install again to confirm the errors are resolved.

For more information, refer to [Managing environment content with a Puppetfile](#) on page 767.

Run a deployment test

You can manually run a full r10k deployment to check your Puppetfile syntax, access to sources, and whether the deployment works through r10k.

The following 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 and doesn't trigger file sync. Only use this for ad-hoc testing. The test deployment command is:

```
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 the command succeeds, the `/etc/puppetlabs/code-staging` directory is populated with directory-based environments and all the necessary modules for every environment.

If the command fails, the error is likely caused by Code Manager's r10k-related settings. The error messages indicate which settings are failing. For more information, refer to [Code Manager settings](#) on page 782.

Source control webhook issues

Refer to [Troubleshoot a Code Manager webhook](#) on page 803.

Monitor /v1/deploy logs

If you're experiencing errors with deployments triggered through the [POST /v1/deploy](#) on page 808 webhook, you can monitor logs when you call the endpoint.

To do this, you'll need to call the [POST /v1/deploy](#) on page 808 endpoint with the `wait` parameter while monitoring the console services log. To monitor the `console-services.log` file, open a terminal window and run:

```
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 might time out before deployment is complete.

If your deployments are timing out too soon, increase one or more of these settings:

- `timeouts_deploy`
- `timeouts_shutdown`
- `timeouts_sync`
- `timeouts_wait`

For descriptions of these settings, refer to [Code Manager parameters](#) on page 791. For instructions on configuring these settings, refer to [Customize Code Manager configuration in Hiera](#) on page 785.

Tip: Deployment timeouts can also occur when the file sync client holds code deployments while waiting for long-running plans to finish. To resolve this, you can increase the `timeouts_sync` setting or allow [Running plans alongside code deployments](#) on page 651.

File sync stops when Code Manager tries to deploy code

Code Manager's code deployments involves accessing many small files. If you store your Puppet code on network-attached storage, you might experience poor performance with deployments due to a slow network or problems with back-end hardware.

If you're experiencing performance issues, try to:

- Tune the network to accommodate the many small files.
- 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 available in the PE console.

If your code deployment succeeds, but a class you added isn't available in the console, try these steps one at a time:

- Refresh your browser.
- Run Puppet to refresh classes.
- Verify that the environment directory exists on disk.
- Check your node group's settings to make sure the group has the correct environment assigned. You might need to run Puppet or redeploy environments after changing environment settings.

Code Manager API

You can use the Code Manager API to deploy code and check the status of deployments on your primary server and compilers without direct shell access.

Forming Code Manager API requests

The Code Manager API accepts well-formed HTTPS requests and requires authentication.

Requests must include a URI path following the pattern:

```
https://<DNS>:8170/code-manager/v1/<ENDPOINT>
```

The variable path components derive from:

- **DNS:** Your primary server's DNS name. You can use `localhost`, manually enter the DNS name, or use a `puppet config print server` command (as explained in [Using example commands](#) on page 28).
- **ENDPOINT:** One or more sections specifying the endpoint, either `deploys`, `webhook`, or `deploys/status`.

Tip: If your Code Manager service does not use port 8170, you need to change the port number in the path.

For example, you could use any of these paths to call the [GET /v1/deploys/status](#) on page 814 endpoint:

```
https://$(puppet config print server):8170/code-manager/v1/deploys/status
https://localhost:8170/code-manager/v1/deploys/status
https://puppet.example.dns:8170/code-manager/v1/deploys/status
```

To form a complete curl command, you need to provide appropriate curl arguments, authentication, and you might need to supply the content type and/or additional parameters specific to the endpoint you are calling.

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

Code Manager API authentication

Code Manager API requests require token-based authentication. For instructions on generating, configuring, revoking, and deleting authentication tokens in PE, go to [Request an authentication token for deployments](#) on page 77 or [Token-based authentication](#) on page 303.

To provide tokens for deploys endpoints, you can use an X-authentication header with the `puppet-access show` command, such as:

```
auth_header="X-Authentication: $(puppet-access show)"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/
status"

curl --insecure --header "$auth_header" "$uri"
```

Or you can use the actual token, such as:

```
auth_header="X-Authentication: <TOKEN>"
uri="https://$(puppet config print server):8170/code-manager/v1/deploys/
status"

curl --insecure --header "$auth_header" "$uri"
```

Important: Unlike the deploys endpoints, when calling the webhook endpoint, you must append the token as a query parameter. Tokens supplied in query parameters might appear in access logs.

Related information

[API index](#) on page 33

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 77

To securely deploy your code, request an authentication token for the deployment user.

POST /v1/deploys

Trigger Code Manager to deploy code to a specific environment or all environments, or use the `dry-run` parameter to test your control repo connection.

Request format

When forming [Code Manager API](#) on page 807 requests to this endpoint, the content type is `application/json`. The body must be a JSON object using the keys described in the following table. You must supply either `deploy-all` or `environments`, and, although not required, you might find the other keys useful in certain situations.

Key	Format	Definition
<code>deploy-all</code>	Boolean	<p>Set to <code>true</code> if you want to trigger code deployments for all known environments.</p> <p>If <code>false</code> or omitted, you must include the <code>environments</code> key.</p> <p>For information about how Code Manager detects environments, refer to Add an environment on page 767.</p>

Key	Format	Definition
environments	Array of strings	Specify the names of one or more specific environments for which you want to trigger code deployments.
		This key is required if <code>deploy-all</code> is <code>false</code> or omitted.
deploy-modules	Boolean	Indicate whether Code Manager deploys modules declared in an environment's Puppetfile.
		If <code>false</code> , modules aren't deployed. If omitted, the default value is <code>true</code> .
		<p>Restriction: Modules are always deployed the first time an environment is deployed, even if you set <code>deploy-modules</code> to <code>false</code>. This ensures environments are fully populated upon first use. If you want to exclude a module from an environments initial deployment, remove or comment-out the module in the environment's Puppetfile.</p>
		For more information, refer to: Managing modules with a Puppetfile on page 768
modules	JSON object	A comma-separated or space-separated list of specific modules to deploy.
wait	Boolean	Indicates how soon you want Code Manager to return a response.
		If <code>false</code> or omitted, Code Manager returns a list of queued deployments immediately after receiving the request.
		If <code>true</code> , Code Manager returns a more detailed response after all deployments have finished (either successfully or with an error).

Key	Format	Definition
dry-run	Boolean	<p>Use to test Code Manager's connection to your source control provider.</p> <p>If <code>true</code>, Code Manager attempt to connect to each of your remotes, attempts to fetch a list of environments from each source, and reports any connection errors.</p> <p>For more information about having multiple remotes, refer to: How the control repository works on page 763</p>

Here are three examples of complete curl commands for the `deploys` endpoint.

This example deploys the production environment and uses the `wait` key to get a more detailed response after the deployment finishes:

```
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 --header "$type_header" --header "$auth_header" --request POST "$uri"
--data "$data"
```

This example deploys two environments and uses 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"
```

This example deploys all environments and returns queueing information immediately after the submitting the request:

```
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

Note: If your request included the `wait` key, the response arrives after the deployments finish. You might have to wait several minutes, depending on the number of environments, deployment sizes, and how geographically dispersed the deployments are.

A successful response contains a list of objects, where each object contains data about a queued or deployed environment. Response objects use keys described in the following table. Which keys are included and possible values for the `status` key depend on the value of the `wait` key in the request.

Key	Definition
<code>environment</code>	The name of the queued or deployed environment.
<code>id</code>	An integer generated by Code Manager that identifies the environment's order in the deployment queue.
<code>status</code>	<p>A code deployment's status at the time of the response.</p> <p>If the request omitted <code>wait</code> or included <code>"wait": false</code>, then the <code>status</code> is either <code>new</code> or <code>queued</code>.</p> <ul style="list-style-type: none"> <code>new</code>: The deploy request is accepted but not yet queued. <code>queued</code>: The deploy is queued and waiting to start. <p>If the request included <code>"wait": true</code>, the <code>status</code> is either <code>complete</code> or <code>failed</code>.</p> <ul style="list-style-type: none"> <code>complete</code>: The deploy is complete and synced to the live code directory on the primary server and compilers. <code>failed</code>: The deploy failed. Response objects for failed deployments also include the <code>error</code> key.
<code>deploy-signature</code>	The commit SHA from the control repo that Code Manager used for the environment's code deploy.
<code>file-sync</code>	<p>Only included if the request included <code>"wait": true</code>.</p> <p>An object containing <code>environment-commit</code> and <code>code-commit</code>, which are commit SHAs used internally by file sync to identify the code synced to the code staging directory.</p> <p>Only included if the request included <code>"wait": true</code>.</p>

Key	Definition
error	<p>Only included if the request included "wait": true and the deployment failed.</p> <p>The error key is an object that uses the following keys to describe the failure:</p> <ul style="list-style-type: none"> • details: Can contain the corrected-name of the environment. • kind: The type of error encountered. • msg: A longer error message that can help you troubleshoot the failure. <p>Failure information remains in this endpoint's responses until cleaned up by garbage collection, even if the environment has a successful deployment after the failure. For more information, refer to Configuring garbage collection on page 787.</p>

For example, this response resulted from a request that included a false or omitted wait key:

```
[
  {
    "environment": "production",
    "id": 1,
    "status": "queued"
  },
  {
    "environment": "testing",
    "id": 2,
    "status": "queued"
  }
]
```

Whereas this example response is from a request that included "wait": true:

```
[
  {
    "deploy-signature": "482f8d3adc76b5197306c5d4c8aa32aa8315694b",
    "file-sync": {
      "environment-commit": "6939889b679fdb1449545c44f26aa06174d25c21",
      "code-commit": "ce5f7158615759151f77391c7b2b8b497aaebce1"
    },
    "environment": "production",
    "id": 3,
    "Code Manager": "Code Manager",
    "status": "complete"
  }
]
```

This example describes a failed deployment:

```
{
  "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/puffpetlabs-apache\".\n"
    },
    "id": 52,
    "status": "failed"
  }
}

```

Tip: If deployments are failing when triggered by the `deploys` endpoint, refer to [Troubleshooting Code Manager](#) on page 804 for information about monitoring logs associated with this endpoint.

POST /v1/webhook

Deploy code by triggering your Code Manager webhook.

Request format

Unlike other POST requests, when forming [Code Manager API](#) on page 807 requests to the `webhook` endpoint, you must append parameters to the URI path, rather than using a JSON body. Available parameters include:

- `type`: Always required. Identifies your Git host.
- `prefix`: Required if your source configuration uses prefixing. Specifies the prefix to use when converting branch names to environment names.
- `token`: Required unless you disabled `authenticate_webhook` in your Code Manager configuration. You must supply the authentication token in the `token` parameter. Tokens supplied in query parameters might appear in access logs.

For more information and examples for each parameter, refer to [Code Manager webhook query parameters](#) on page 802.

For example, this request includes the `type` and `token` parameters:

```
curl -X POST 'https://$(puppet config print server):8170/code-manager/v1/webhook?type=github&token=<TOKEN>'
```

Tip: For more information, refer to [Triggering Code Manager with a webhook](#) on page 801.

Response format

When your Code Manager webhook is automatically triggered by a push to the control repo, all responses appear in your Git provider's interface. Code Manager does not give command line responses to automatic webhook triggers.

If you use a curl command to manually trigger the webhook, and your request is well-formed and valid, Code Manager returns an OK response. This only indicates that the request was valid, it does not indicate whether a resulting code deployment succeeded or failed.

Error responses

If an error occurs when the webhook is automatically triggered, check your Git provider interface for error responses. If there is a problem with the webhook, refer to [Troubleshoot a Code Manager webhook](#) on page 803.

If you use a curl command to manually trigger the webhook, and the request has a missing or invalid `type`, the endpoint returns an `unrecognized-webhook-type` error along with a copy of the supplied `type` value and a list of valid `type` values.

GET /v1/deploys/status

Get the status of code deployments that Code Manager is currently processing for each environment. You can specify an `id` query parameter to get the status of a particular deployment.

Request format

When forming [Code Manager API](#) on page 807 requests to this endpoint, the request is a basic call with authentication, such as:

```
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 --header "$type_header" --header "$auth_header" "$uri"
```

A basic request returns the status of all deployments in the queue. You can append the optional `id` query parameter to get the status of a specific deployment by calling its position in the current deployment queue. For example, this request calls the status of the fifth deployment in the current queue:

```
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=5"

curl --header "$type_header" --header "$auth_header" "$uri"
```

Response format

For requests that **do not** include the `id` parameter, the successful response is a JSON object describing the status of each deployment in the queue. The body is divided into three secondary objects (`deploys-status`, `file-sync-storage-status`, and `file-sync-client-status`) that report information about deployments depending on the deployments' statuses.:

- **deploys-status**: Contains four arrays representing possible code deployment statuses, which are `new`, `queued`, `deploying`, or `failed`.
 - For each new, queued, deploying, and failed deployment, there is a nested object containing the deployment's position in the queue (`id`), the environment name (`environment`), and the time the deployment was put in the queue (`queued-at`).
 - For failed deployments, there is an additional `error` object describing the failure (`details` and `corrected-name`), the type of error (`kind`), and a longer error message (`msg`). Failure information remains in this endpoint's responses until cleaned up by garbage collection, even if the environment has a successful deployment after the failure. For more information, refer to [Configuring garbage collection](#) on page 787.
 - If there are no deployments with a particular status in the queue, the array for that status is empty.
 - Successfully completed deployments are reported in either the `file-sync-storage-status` or `file-sync-client-status` objects.
- **file-sync-storage-status**: Contains a `deployed` object that lists environments that Code Manager has successfully deployed to the code staging directory, but not yet synced to the live code directory. For each deployed environment, the response includes the environment name (`environment`), the deployment date and time (`date`), and the commit SHA from the control repo that Code Manager used for the environment's code deploy (`deploy-signature`).

- **file-sync-client-status**: Lists the status of 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 the live code directory. Contains:
 - **all-synced**: A Boolean indicating whether all requested code deployments are synced to the live code directories on their respective servers.
 - **file-sync-clients**: An object containing a list of servers that Code Manager deployed code to. For each server, the response includes the date and time of the server's last check in (`last_check_in_time`), whether the server synchronized with file sync storage (`synced-with-file-sync-storage`), and an array of deployment objects associated with that server. Each deployment object includes the environment name (`environment`), the deployment date and time (`date`), and the commit SHA from the control repo that Code Manager used for the environment's code deploy (`deploy-signature`).

For requests that include the `id` parameter, the response is condensed and uses the following keys to describe the specified deployment:

- **environment**: The name of the environment.
- **status**: The deployment's status at the time of the response. Can be new, queued, deploying, failed, syncing, or synced.
- **queued-at**: The date and time when the deployment was put in the queue.

For example, this is a response for a single deployment:

```
{
  "environment": "production",
  "id": 1,
  "status": "deploying",
  "queued-at": "2018-05-10T21:44:25.000Z"
}
```

And this is an example response for an entire deployment queue:

```
{
  "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": [
      {
        "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/puffpetlabs-apache\".\n"
    },
    "queued-at": "2018-06-01T21:28:18.292Z"
}
],
},
"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"
                    }
                ]
            }
        }
    }
}

```

```

    }
}
```

About file sync

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

When a code deployment is triggered, Code Manager uses r10k to fetch code (over HTTPS or SSH) from the Git server and places the code into the staging directory on the primary server (at `/etc/puppetlabs/code-staging`), which is an internal Git server.

Next, the file sync storage service on the primary server detects the change in the staging directory, and the file sync clients pause Puppet Server to avoid conflicts during synchronization. Finally, the file sync clients synchronize the new code to the live code directories on the primary server and compilers (usually at `/etc/puppetlabs/code`).

File sync only deploys Puppet code when an agent is ready to receive the new code. This ensures that your agents' code doesn't change during a Puppet run. File sync triggers an environment cache flush when the deployment is finished, which ensures that new Puppet agent runs use the newly-deployed Puppet code.

Important:

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync.

Directly accessing the file sync API is unsupported and not recommended.

For information about how code deployments are triggered, refer to [How Code Manager works](#) on page 775.

Tip: If you want code deployments to run uninterrupted while Orchestrator plans are running, you can enable [Running plans alongside code deployments](#) on page 651.

File sync terms

Understanding these terms is helpful for understanding file sync.

Control repository

Used for storing your Puppet code and maintaining separate code for different environments (such as production, development, or testing). Each environment is represented by a branch of the control repo. For more information, refer to [Managing environments with a control repository](#) on page 763.

Each environment branch has a Puppetfile specifying exactly which modules to install in each environment. You can learn more [About Environments](#) in the Puppet documentation.

Live code directory

This directory contains all your Puppet manifests and modules for each environment (Code Manager creates directory environments based on the branches you've set up in your control repo). Puppet Server uses this directory for catalog compilation. This directory is present on your primary server and all compilers.

The directory's location is specified in the Puppet Server `master-code-dir` setting and the `Puppet$codedir` setting. The default value is `/etc/puppetlabs/code`, which is also the default location for the live code directory.

In file sync configuration settings, the live code directory is sometimes abbreviated as `live-dir`.

Staging code directory

Code Manager detects code changes that you make in your control repository, and then Code Manager stages the updated code in the *staging code directory*. From there, file sync moves the updated code to the live code directories.

While the primary server and all compilers have their own live code directories, the staging code directory is only present on the primary server. The default location is `/etc/puppetlabs/code-staging`.

How file sync works

File sync helps distribute your Puppet code to your primary server, compilers, and agents.

By default, file sync is disabled and there is no staging directory on the primary server. If you're upgrading from Puppet Enterprise (PE) 2015.2 or earlier, file sync is automatically disabled after the upgrade. When you [Set up Code Manager](#) on page 778, you also enable file sync. This creates the staging directory on the primary server, which Code Manager can then populate with Puppet code it detects in your control repo.

Once Code Manager pulls code from your control repo and places it in the staging directory, file sync synchronizes the code to the primary server's live code directory, and then to the live code directories on all compilers. Once deployed to compilers, the new code is available to agents that check in to those compilers.

Usually, code deployments are handled by Code Manager, but, only if absolutely necessary, you could use the POST `file-sync/v1/commit` endpoint to manually trigger a file sync deployment.

Important:

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync.

File sync endpoint documentation is for informational purposes only. Directly accessing the file sync API is unsupported and not recommended.

You can call the endpoint from the primary server, and your request must contain:

- A content type specification in the header: `Content-Type: application/json`
- Authentication. Namely, the primary server's SSL certificate and private key, and the Puppet CA's certificate.
- The POST action.
- A URI made up of:
 - The primary server's fully qualified domain name, which you can call with `puppet config print server` or the literal domain name.
 - The port the endpoint listens on. The default is 8140.
 - The endpoint identifier: `file-sync/v1/commit`
- A JSON body containing `{"commit-all": true}`

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"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

A successful request triggers file sync to commit the latest changes from the staging directory to the primary server's live code directory. The next time the compilers poll the file sync service for code changes, they receive the newly-committed code and deploy it into their own live code directories, where it is available for agents checking in to those compilers. By default, compilers poll for changes every 5 seconds.

Code commits can be restricted to a specific environment and can include details, such as a message or 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 Code Manager. Because Code Manager is disabled by default, file sync is also disabled by default. To enable file sync, you must [Set up Code Manager](#) on page 778. You can enable and configure Code Manager either during or after you install Puppet Enterprise (PE).

In the PE console, the `file_sync_enabled` parameter, in the `puppet_enterprise::profile::master` class, defaults to `automatic`, which means that file sync is enabled and disabled automatically when Code Manager is enabled or disabled. If you set the `file_sync_enabled` 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 `puppet_enterprise::profile::master` class definitions – you must add the parameter to the class if you want to set it.

Resetting file sync

You'll need to reset the file sync service if file sync enters a failure state, if file sync consumes all available disk space, or a repository becomes irreparably corrupted.

Resetting the file sync service deletes the commit history for all repositories that file sync tracks. This frees up disk space and returns the service to a fresh state while preserving code in the staging directory.

1. On your primary server:

- **If you use Code Manager:** Make sure the code you want to deploy is present in the staging directory at `/etc/puppetlabs/code-staging`, and make sure that your most-recently deployed code is present in your control repository (so you can re-sync it).
- **If you use r10k:** Trigger and r10k code deployment, and make sure your most-recently deployed code is present in your control repository (so you can re-sync it).
- **If you use file sync by itself:** Make sure the code you want to deploy is present in the staging directory at `/etc/puppetlabs/code-staging`.

2. Stop the Puppet Server service by running:

```
puppet resource service pe-puppetserver ensure=stopped
```

3. Delete the data directory located at:

```
/opt/puppetlabs/server/data/puppetserver/filesync/storage
```

4. Restart the Puppet Server service by running:

```
puppet resource service pe-puppetserver ensure=running
```

5. Take the appropriate action to redeploy your code:

- **If you use Code Manager:** Deploy your code to all environments by [Triggering Code Manager on the command line](#) on page 795.
- **If you use r10k or standalone file sync:** Perform a commit.

When you reset the file sync service, it creates fresh repositories on each client and on the storage server for the code it manages.

Checking your deployments

If necessary, you can retrieve information about file sync deployments by calling the `status/v1/services/file-sync-storage-service` endpoint.

Important:

File sync is part of Code Manager, and you usually don't need to directly configure or trigger file sync.

File sync endpoint documentation is for informational purposes only. Directly accessing the file sync API is unsupported and not recommended.

To call the `status/v1/services/file-sync-storage-service` endpoint, use this curl command:

```
curl --insecure "https://$(puppet config print server):8140/status/v1/
services/file-sync-storage-service?level=debug"
```

For general information about forming curl commands, authentication in commands, and Windows modifications, go to [Using example commands](#) on page 28.

A successful request returns a JSON-formatted list of:

- All clients that file sync is aware of.
- When those clients last checked in.
- Which commit the clients currently have deployed.

Tip: For pretty printing, pipe the response to: `python -m json.tool`

To check if a specific commit was deployed, review the `latest_commit` in the response. The `latest_commit` SHA, in this endpoint's response, is specific to file sync. This SHA does not correspond to a commit from your control repository.

File sync cautions

Keep these warnings in mind when working with file sync.

File sync API

Because file sync is part of Code Manager, Code Manager handles communication with the file sync API. Information about the file sync API in this documentation is for informational purposes only.

Important: Do not directly access the file sync API.

Where to edit code

With Code Manager, you only modify code in your control repo. Changes made in invalid locations are overwritten by the next deployment. For more information refer to [Understanding file sync and the staging directory](#) on page 776.

While extremely uncommon, if you're using file sync without Code Manager, only modify your Puppet code in the staging directory.

By default, the `enable-forceful-sync` parameter is set to `true` in Puppet Enterprise (PE). If this is set to `false`, file sync no longer overwrites changes in the live code directory. Instead, it logs errors to the Puppet Server log (at `/var/log/puppetlabs/puppetserver/puppetserver.log`).

If you need to set this parameter to `false`, you must add it with Hiera:

```
puppet_enterprise::master::file_sync::file_sync_enable_forceful_sync: false
```

The `puppet module` command and file sync

The `puppet module` command doesn't work with file sync. You'll need to specify modules in your environment's Puppetfiles and use Code Manager to handle module code deployments. For information and instructions refer to:

- [Managing modules with a Puppetfile](#) on page 768
- [Managing code with Code Manager](#) on page 774

Permissions

File sync runs as the `pe-puppet` user. To sync files, file sync **must** have permission to read the staging directory and to write to all files and directories in the live code directories. You can run the following command to make sure the `pe-puppet` user owns the required directories:

```
chown -R pe-puppet /etc/puppetlabs/code /etc/puppetlabs/code-staging
```

Note: Puppet Enterprise chowns the content of the code directory automatically. If users encounter problems while using `find` and `chown`, they can disable the behavior by setting the following parameter: `puppet_enterprise::master::file_sync::chown_code_to_pe_puppet` to `false` to skip that `find/chowns`.

Environment isolation metadata

File sync generates `.pp` metadata files in your staging code directory and live code directories. These files provide environment isolation for your resource types, which ensures that each environment uses the correct version of the resource type.



CAUTION: Do not delete or modify the metadata files. Do not use expressions from these files in regular manifests.

For more details about these files and how they isolate resource types in multiple environments, refer to [Environment isolation](#) in the Puppet documentation. For information about when these files are generated, refer to [Environment isolation metadata and Code Manager](#) on page 777.

Managing code with r10k

r10k is a code management tool that allows you to manage your environment configurations (such as production, testing, and development) in a source control repository. Unlike Code Manager's automated deployments, r10k requires you to *manually* deploy code changes from your control repository using the r10k command line tool on your primary server and all compilers.

Based on the code in your control repository branches, r10k creates environments on your primary server and also installs and updates the modules you want in each environment.

Tip: Whenever possible, we recommend [Managing code with Code Manager](#) on page 774. Only use r10k if you cannot use Code Manager.

- [Set up r10k](#) on page 822

You must set up r10k to use it as your code management tool.

- [Configure r10k](#) on page 822

To configure r10k in an existing Puppet Enterprise (PE) installation, set r10k parameters in the PE console. You can also use the console to adjust r10k settings.

- [Customizing r10k configuration](#) on page 823

Set parameters in Hiera to customize your r10k configuration.

- [Deploying environments with r10k](#) on page 831

Deploy environments on the command line with the `r10k deploy` command.

- [r10k command reference](#) on page 834

The `r10k` command accepts actions, options, and subcommands.

Set up r10k

You must set up r10k to use it as your code management tool.

Tip: Whenever possible, we recommend [Managing code with Code Manager](#) on page 774, because Code Manager does not require manual code deployment. Only use r10k if you cannot use Code Manager.

To set up r10k, you must:

1. Prepare for [Managing environments with a control repository](#) on page 763. This involves creating a Git control repository that has a Puppetfile.

r10k uses the control repo to maintain and deploy your Puppet code and data. You can also create separate deployment environments in your Puppet infrastructure by creating branches in your control repository (such as a `development` branch for a development environment). r10k tracks your environments and updates them according to the changes you make in your control repo.

The Puppetfile specifies which modules and data to install in your environment, including what versions to install, and where to download the modules or other content.

2. [Configure r10k](#) on page 822.

3. Optional: [Customize your r10k configuration](#) in Hiera.

4. Use the r10k command line tool to [Deploy environments](#).

Related information

[Moving from r10k to Code Manager](#) on page 777

Moving from r10k to Code Manager can improve automation of your code management and deployments.

[Managing environment content with a Puppetfile](#) on page 767

A Puppetfile specifies detailed information about each environment's Puppet code and data.

Configure r10k

To configure r10k in an existing Puppet Enterprise (PE) installation, set r10k parameters in the PE console. You can also use the console to adjust r10k settings.

Before you begin

You need a control repo (with a Puppetfile) and the file path for the SSH private key you created when you set up your control repo. For information and instructions on setting up a control repo, go to [Managing environments with a control repository](#) on page 763.

1. In the PE console, go to **Node groups > PE Master > Classes**, and locate the `puppet_enterprise::profile::master` class.

- For the `r10k_remote` parameter, enter a string that is a valid SSH URL for your Git control repository, such as `git@<YOUR.GIT.SERVER.COM>:puppet/control.git`.

Important: Some Git providers have additional requirements for enabling SSH access. For example, BitBucket requires `ssh://` at the beginning of the SSH URL (such as `ssh://git@<YOUR.GIT.SERVER.COM>:puppet/control.git`). See your provider's documentation for this information.

- For the `r10k_private_key` parameter, enter a string specifying the path to the SSH private key you created when you set up your control repo, such as `/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519`.

This key permits the `pe-puppet` user to access your Git control repo. The private key 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.ed25519`.

- Run Puppet on your primary server and compilers.

You can [customize your r10k configuration](#) in Hiera, if needed.

To [deploy environments with r10k](#), you must use the command line to manually trigger deployments. PE does not automatically run `r10k` after you configure it.

Related information

[Configuration parameters and the pe.conf file](#) on page 129

A `pe.conf` file is a HOCON formatted file that declares parameters and values used to install, upgrade, or configure Puppet Enterprise (PE). A default `pe.conf` file is available in the `conf.d` directory in the installer tarball.

Customizing r10k configuration

Set parameters in Hiera to customize your `r10k` configuration.

The customize your configuration:

- In your control repo, open the `data/common.yaml` file.
- Add parameters to the `pe_r10k` class. Use the following format:

```
pe_r10k::<PARAMETER>: <SETTING>
```

For example, these parameters specify a Git repo cache directory and the location from which to fetch the source repository:

```
pe_r10k::cachedir: /var/cache/r10k
pe_r10k::remote: git://git-server.site/my-org/main-modules
```

Some parameters are described in detail below, along with a list of all [r10k parameters](#) on page 830.

- Run Puppet on the primary server.
- [Deploy environments with r10k](#). PE does not automatically run `r10k` after you configure it.

Related information

[Configure settings with Hiera](#) on page 212

Hiera is hierarchy-based 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 defined, 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 Puppet Enterprise (PE) configuration settings.

Configuring the r10k base directory

The `r10k` base directory specifies the path where environments are created for your control repo.

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.

The `r10k_basedir` parameter accepts a string-formatted path, such as: `/etc/puppetlabs/code/environments`

Important: The `r10k_basedir` setting **must match** the `environmentpath` in your `puppet.conf` file, or Puppet can't access your new directory environments. For details about this setting, refer to [environmentpath in the open source Puppet documentation](#).

If you have multiple source repos, you must specify the `basedir` for each source (in the `sources` parameter) instead of the global `r10k_basedir` setting. Specifying both base directory settings causes errors.

Configuring post-deployment commands

To set commands to run after deployments complete, use the `postrun` parameter.

This parameter accepts the full command as an array of strings, which can be used as an argument vector. You can set this parameter only once. For example:

```
postrun: [ '/usr/bin/curl' , '-F' , 'deploy=done' , 'http://my-app.site/endpoint' ]
```

Configuring purge levels

The `purge_levels` setting, within the `deploy` parameter, controls which unmanaged content `r10k` purges after a deployment.

The `purge_levels` setting accepts an array of strings specifying what content `r10k` purges during code deployments. You can specify one or more of `deployment`, `environment`, `puppetfile`, and `purge_allowlist`.

For example:

```
deploy:
  purge_levels: [ 'deployment' , 'environment' , 'puppetfile' ]
```

The default setting is `['deployment' , 'puppetfile']`.

Each purge level option is explained below.

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.

Important: Removing `deployment` from `purge_levels` allows 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 automatically.

environment

After a given environment is deployed, `r10k` recursively removes content that is:

- Not committed to the control repo branch that maps to that environment.
- Not declared in a Puppetfile committed to that branch.

With the `environment` purge level, `r10k` loads and parses the Puppetfile for the environment, even if the `--modules` flag is not set, so that `r10k` can check whether content is declared in the Puppetfile. However, `r10k` doesn't actually deploy Puppetfile content *unless* the environment is new or the `--modules` flag is set.

If `r10k` encounters an error while evaluating the Puppetfile or deploying its contents, no environment-level content is purged.

puppetfile

After Puppetfile content for a given environment is deployed, r10k recursively removes content in any directory managed by the Puppetfile, if that content is not declared in the Puppetfile.

Directories managed by a Puppetfile include the configured `moduledir` (which defaults to `modules`), as well as any alternate directories specified as an `install_path` option to any Puppetfile content declarations.

purge_allowlist

The `purge_allowlist` setting exempts the specified filename patterns from being purged. This setting affects only environment purging. The value for this setting must be a list of shell-style filename patterns formatted as strings.

See the [Ruby documentation](#) about the `fnmatch` method for information on valid patterns. Both the `FNM_PATHNAME` and `FNM_DOTMATCH` flags are in effect when r10k considers the allowlist.

Patterns are relative to the root of the environment being purged and, by default, **do not match recursively**. For example, an allowlist value of `*myfile*` preserves only matching files at the root of the environment. To preserve matching files throughout the deployed environment, you need to use a recursive pattern such as `**/*myfile*`.

Files matching an allowlist pattern might still get removed if they exist in a folder that is otherwise subject to purging. In this case, use an additional allowlist rule to preserve the containing folder, for example:

```
deploy:
  purge_allowlist: [ 'custom.json', '**/*.xpp' ]
```

Configuring Forge settings

To configure how r10k downloads modules from the Forge, specify the `forge_settings` parameters in Hiera.

This parameter specifies where Forge modules are installed from, and it sets a proxy for all Forge interactions. The `forge_settings` parameter accepts a hash that can use the following keys:

- `baseurl`: Indicate where Forge modules are installed from. The default is `https://forgeapi.puppetlabs.com`.
- `authorization_token`: Specify the token for authenticating to a custom Forge server.
- `proxy`: Set the proxy for all Forge interactions.

For example, this configuration specifies a custom Forge server that doesn't require authentication:

```
pe_r10k::forge_settings:
  baseurl: 'https://private-forge.mysite'
```

If your custom Forge server requires authentication, you must specify both `baseurl` and `authorization_token`. You must format `authorization_token` as a string prepended with `Bearer`, particularly if you use Artifactory as your Forge server. For example:

```
pe_r10k::forge_settings:
  baseurl: 'https://private-forge.example'
  authorization_token: 'Bearer <TOKEN>'
```

The `proxy` parameter sets a proxy for all Forge interactions. This setting overrides the global `proxy` setting but only for Forge operations (refer to the global `proxy` setting for more information). You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example:

```
pe_r10k::forge_settings:
  proxy: 'http://proxy.example.com:3128'
```

Tip: If you set a global `proxy`, but you don't want Forge operations to use a proxy, under the `forge_settings` parameter, set `proxy` to an empty string.

Configuring Git settings

To configure r10k to use a specific Git provider, a private key, a proxy, or multiple Git source repositories, specify the `git_settings` parameter.

The `r10k git_settings` parameter accepts a hash that can use the `private_key`, `provider`, `proxy`, `username`, and `repositories` keys. For example:

```
pe_r10k::git_settings:
  provider: "rugged"
  private_key: "/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519"
  username: "git"
```

`private_key`

The `private-key` setting is required, and, if it is not specified, it gets a default value from the `puppet_enterprise::profile::master` class.

Use `private-key` to specify the path to the file containing the default private key that you want Code Manager to use to access control repos, for example:

```
/etc/puppetlabs/puppetserver/ssh/id-control_repo.ed25519
```

Important: The `pe-puppet` user must have read permissions for the private key file, and the SSH key can't require a password.

`provider`

Allows r10k to interact with Git repositories using multiple Git providers. Valid values are `rugged` and `shellgit`.

For more information about this setting, refer to the [r10k documentation](#) on GitHub.

`proxy`

The `proxy` key sets a proxy specifically for Git operations that use an HTTP(S) transport. This setting overrides the global `proxy` setting but only for Git operations (For more information, refer to the global `proxy` setting). You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example:

```
proxy: 'http://proxy.example.com:3128'
```

Tip:

To set a proxy for only one specific Git repository (or when you have multiple control repos), set `proxy` within the `repositories` key.

If you set a global `proxy`, but you don't want Git operations to use a proxy, under the `git_settings` parameter, set `proxy` to an empty string.

`username`

If the Git remote URL does not provide a username, supply the relevant `username` as a string.

`repositories`

The `repositories` key specifies a list of repositories and their respective private keys or proxies. Use `repositories` if:

- You need to configure different proxy settings for specific repos, instead of all Git operations.

- You have multiple control repos.

Important: If you have multiple control repos, the `sources` setting and the `repositories` setting must match.

The `repositories` setting accepts a hash that uses the `remote`, `private-key`, `proxy`, and `username` keys.

The `remote` key specifies the repository to which the subsequent `private-key`, `username`, or `proxy` settings apply. The `private-key`, `username`, and `proxy` settings have the same requirements and functions as described above, except that, when inside `repositories`, these settings only apply to a single repository.

For example, this `repositories` hash specifies a unique private key for one repo and a unique proxy for another repo:

```
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"
```

Tip: If you set a global proxy or a `git_settings` proxy, but you don't want a specific repo to use a proxy, in the `repositories` hash, set that specific repo's `proxy` to an empty string.

Configuring proxies

If you need r10k to use a proxy connection, use the `proxy` parameter. You can set a global proxy for all HTTP(S) operations, proxies for Git or Forge operations, or proxies for individual Git repositories.

Where you specify the `proxy` parameter depends on how you want to apply the setting:

- To set a proxy for all r10k operations occurring over an HTTP(S) transport, set the global `proxy` setting.

Tip: If you don't supply a global `proxy`, but you have defined a proxy in an environment variable, r10k uses the value from the highest-ranking `*_proxy` environment variable as the global r10k `proxy`. In order of precedence, r10k looks for `HTTPS_PROXY`, then `https_proxy`, then `HTTP_PROXY`, and finally `http_proxy`. If you have defined neither a global `proxy` nor any `*_proxy` environment variables, the global `proxy` setting defaults to no proxy.

- To set proxies only for Git operations or individual Git repos, set the appropriate `proxy` key under the `git_settings` parameter.
- To set a proxy only for Forge operations, set the `proxy` key under the `forge_settings` parameter.

You can set an unauthenticated proxy or an authenticated proxy with either Basic or Digest authentication. For example, this setting is for an unauthenticated proxy:

```
proxy: 'http://proxy.example.com:3128'
```

Whereas this setting is for a password-authenticated proxy:

```
proxy: 'http://user:password@proxy.example.com:3128'
```

Override proxy settings

You can override the global `proxy` setting if you want to:

- Set a different proxy setting for Git or Forge operations.
- Specify a different proxy setting for an individual Git repo.
- Specify a mix of proxy and non-proxy connections.

To override the global proxy setting for all Git or Forge operations, you need to set the `proxy` key under the `git_settings` or `forge_settings` parameters.

To set a proxy for an individual Git repository (or if you have multiple control repos), set the `proxy` key in the `repositories` hash under the `git_settings` parameter.

If you have set a global, Git, or Forge proxy, but you **don't** want a certain setting to use any proxy, set the `proxy` parameter to an empty string. For example, if you set a global proxy, but you don't want Forge operations to use a proxy, you would specify an empty string under the `forge_settings` parameter, such as:

```
puppet_enterprise::master::code_manager::forge_settings:
  proxy: ''
```

Proxy server logging

If r10k uses proxy server during a deployment, r10k logs the server at the debug log level.

Configuring sources

If you are managing multiple control repos with r10k, you must use the `sources` parameter to specify a map of your source repositories.

The `sources` parameter is necessary when r10k is managing multiple control repos. For example, your Puppet environments are in one control repo and your Hiera data is in a separate control repo.

Important:

The `sources` setting and the `repositories` setting (under `git_settings`) must match.

If `sources` is set, you can't use r10k's global `remote` and `r10k_basedir` settings.

The `sources` parameter consists of a list of source names along with a hash that can contain the `remote`, `basedir`, `prefix`, `ignore_branch_prefixes`, and `invalid_branches` key for each source. For example:

```
myorg:
  remote: "git://git-server.site/myorg/main-modules"
  basedir: "/etc/puppetlabs/puppet/environments"
  prefix: true
  ignore_branch_prefixes:
    - "doc"
  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

The `remote` parameter specifies the location from which to fetch the source repo. r10k must be able to fetch the remote without any interactive input. This means fetching the source can't require inputting a user name or password. You must supply a valid URL, as a string, that r10k can use to clone the repo, such as: `"git://git-server.site/myorg/main-modules"`

Tip: If your sources are in different Git providers, you might need to configure the `providers` parameter in the [r10k Git settings](#).

basedir

Specifies the path to the location where this source's environments are created. This directory is entirely managed by r10k, and any contents that r10k did not put there are removed.

Important: The `basedir` setting **must match** the `environmentpath` in your `puppet.conf` file, or Puppet can't access your new directory environments.

If you specify `basedir` in sources, **do not** also specify the global `r10k_basedir` setting. Specifying both base directory settings causes errors.

prefix

The `prefix` parameter specifies a string to use as a prefix for the names of environments derived from the specified source. Set this to a specific string if you want to use a specific prefix, such as "testing". Set this to `true` to use the source's name as the prefix. The `prefix` parameter prevents collisions (and confusion) when multiple sources with identical branch names are deployed into the same directory.

For example, the following settings might cause errors or confusion because there would be two `main-modules` environments deployed to the same base directory:

```
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: true
  invalid_branches: 'correct_and_warn'
```

However, by changing one `prefix` to "testing", the two environments become more distinct, since the directory would now have a `myorg-main-modules` environment and a `testing-main-modules` environment:

```
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'
```

ignore_branch_prefixes

Use `ignore_branch_prefixes` if you want r10k to **not** deploy some branches in a specified source. If you omit this parameter, then r10k attempts to deploy all branches.

Specify prefixes you want r10k to ignore as a list of strings, such as:

```
sources:
  mysource:
    remote: "git://git-server.site/mysource/main-modules"
    basedir: "/etc/puppet/environments"
    ignore_branch_prefixes:
      - "test"
      - "dev"
```

When r10k runs, if the beginning of a branch's name matches one of the supplied prefixes, r10k ignores the branch and does not deploy an environment based on that branch.

For example, ignoring the prefix "test" ignores branches starting with test, which could be test as a complete branch name, or test followed by any amount or variation of characters, such as test*, testing*, tester*, test_*, and so on.

Tip: Your ignore_branch_prefixes strings are inherently followed by a wildcard. For example, "test" is inherently treated like test*. Do not include wildcard characters in your prefix strings, because r10k interprets them as being literally part of the branch names.

This setting is useful for ignoring branches named after support tickets, training branches, documentation branches, or other such branches that you don't want r10k to try to deploy as environments. If you want to ignore a particular branch without excluding other similarly-prefixed branches, supply the branch's full name in the ignore_branch_prefixes list.

invalid_branches

Specifies how you want r10k to handle branch names that can't cleanly map to Puppet environment names. Supply one of the following strings:

- "error": Ignore branches that have non-word characters, and report an error about the invalid branches.
- "correct": Without providing a warning, replace non-word characters with underscores.
- "correct_and_warn": Replace non-word characters with underscores, and report a warning about the altered branch names. This is the default value if omitted.

Locking r10k deployments

The deploy: write_lock setting allows you to temporarily disallow r10k code deploys without completely removing the r10k configuration.

This setting is useful for preventing r10k deployments at certain times, or for preventing deployments from interfering with a common set of code that might be touched by multiple r10k configurations.

To enable this, add the write_lock setting under the deploy parameter and supply a message that is returned when someone attempts to deploy code. For example:

```
deploy:
  write_lock: "Deploying code is disallowed until the next maintenance
  window."
```

r10k parameters

The following parameters are available for r10k. Parameters are optional unless otherwise stated.

Parameter	Description	Type	Default value
cachedir	The file path to the location where r10k caches Git repositories.	String	/var/cache/r10k
deploy	For Configuring purge levels on page 824 and Locking r10k deployments on page 830.	Hash	<ul style="list-style-type: none"> • purge_level: ['deployment', 'puppetfile'] • write_lock: Omitted
forge_settings	For Configuring Forge settings on page 825.	Hash	No default.

Parameter	Description	Type	Default value
git_settings	For Configuring Git settings on page 826.	Hash	Can use the default private-key value set in console. Otherwise, there are no default settings.
proxy	For Configuring proxies on page 827. Can be global (all HTTP(s) transports) or part of the git_settings or forge_settings hashes.	An empty string or a string indicating a proxy server (with or without authentication)	The global proxy can use a *_proxy environment variable, if one is set. Otherwise, there are no defaults.
postrun	For Configuring post-deployment commands on page 824.	Array of strings to use as an argument vector	No default.
remote	A valid SSH URL specifying the location of your Git control repository, if you have only one control repo. If you have multiple Git repos, specify sources instead of remote.	String	If r10k_remote is specified in the puppet_enterprise::profile::m
r10k_basedir	For Configuring the r10k base directory on page 823, if you have only one control repo. If you have multiple Git repos, specify basedir in sources instead.	String	No default, but must match the environmentpath in your puppet.conf file.
sources	For Configuring sources on page 828 if you have multiple control repos.	Hash	No default.

Deploying environments with r10k

Deploy environments on the command line with the r10k deploy command.

The first time you run r10k, deploy all environments and modules by running:

```
r10k deploy environment
```

This command:

1. Checks your control repository to see which branches are present.
2. Maps those branches to the Puppet directory environments.
3. Clones your Git repo and either creates (if this is your first run) or updates (if it is a subsequent run) your directory environments with the contents of your repo branches.

Restriction: When running commands to deploy code on your primary server, r10k needs write access to your environment path. You need to run r10k as the `pe-puppet` user, as root, or use `sudo`. Running as root requires access control to the root user.

Related information

[r10k command reference](#) on page 834

The `r10k` command accepts actions, options, and subcommands.

Updating environments

To update environments with `r10k`, use the `r10k deploy environment` command.

Update all environments

The `r10k deploy environment` command updates all existing environments and recursively creates new environments.

From the command line, run: `r10k deploy environment`

Structured in this way, this command updates modules only on the environment's first deployment. On subsequent updates, it only updates the environment.

Update all environments and modules

Add the `--modules` flag to the `r10k deploy environment` command to update all environments and their modules.

From the command line, run: `r10k deploy environment --modules`

This command:

- Updates all existing environments.
- Creates new environments, if any new branches are detected.
- Deletes old environments, if any branches no longer exist.
- Recursively updates all environment modules declared in each environment's Puppetfile.

This command does the maximum possible deployment work. Therefore it is the slowest method for `r10k` deployments. Usually, you'll use the less resource-intensive commands for updating specific environments and modules.

Update a single environment

To update a single environment, specify the environment name with the `r10k deploy environment` command.

From the command line, run:

```
r10k deploy environment <ENVIRONMENT_NAME>
```

Formatted in this way, this command updates one environment. It deploys modules only during the environment's first deployment. On subsequent deployments, it only updates the environment.

If you are actively developing a specific environment, this command is the quickest way to deploy your changes so you can test them.

Update a single environment and its modules

To update a specific environment's content and its modules, add the `--modules` flag to your environment-specific command. This is useful if you want to make sure that a given environment is fully up to date and has modules recently declared in the environment's Puppetfile.

On the command line, run:

```
r10k deploy environment <ENVIRONMENT_NAME> --modules
```

Installing and updating modules

The `r10k deploy` module command installs or updates the modules specified in each environment's Puppetfile.

Update specific modules across all environments

To update specific modules across all environments, specify the modules with the `r10k deploy` module command.

Important: Before updating modules, you must [Update all environments and modules](#) on page 832.

To update one module across all environments, append the module name to the command, such as:

```
r10k deploy module <MODULE_NAME>
```

To update multiple modules across all environments, append the module names to the command, separated by spaces. For example, this command updates the `apache`, `jenkins`, and `java` modules:

```
r10k deploy module apache jenkins java
```

If a specified module is not declared in an environment's Puppetfile, that environment is skipped.

Update one or more modules in a single environment

To update specific modules in a specific environment, specify both the environment and the modules in your command.

Important: Before updating modules, you must [Update a single environment and its modules](#) on page 832.

The command format is:

```
r10k deploy module -e <ENVIRONMENT_NAME> <MODULE_NAME>
```

The first argument supplied after `-e` is interpreted as an environment name. Anything after this is treated as a module name. You can append multiple module names, but only one environment name.

For example, this command updates the `apache`, `jenkins`, and `java` modules in the `production` environment:

```
r10k deploy module -e production apache jenkins java
```

If the specified module is not described in a given environment's Puppetfile, that module is skipped.

Get environment details with r10k

The `r10k deploy display` command returns information about your environments and modules. This subcommand does not deploy environments, it only displays information about the environments and modules `r10k` is managing.

This command can return various levels of detail about the environments:

- To get information about all environments `r10k` manages, run:

```
r10k deploy display
```

- To get information about all managed environments and modules declared in their Puppetfiles, append the Puppetfile flag (`-m`). For example:

```
r10k deploy display -m
```

- To get expected and actual versions of modules in all environments, append the `-m` and `--detail` flags. For example:

```
r10k deploy display -m --detail
```

- To get expected and actual versions of modules in specific environments, append the `-m` and `--detail` flags along with one or more environment names, such as:

```
r10k deploy display -m --detail <ENVIRONMENT_NAME> <ENVIRONMENT_NAME>
```

r10k command reference

The `r10k` command accepts actions, options, and subcommands.

Use this format for `r10k` commands:

```
r10k <ACTION> <ACTION_OPTIONS> <SUBCOMMANDS>
```

r10k command actions

The `r10k` command supports these actions and action options.

Action	Description
deploy	Performs a specified subcommand operation on environments. Accepts r10k deploy subcommands on page 835.
help	Returns <code>r10k</code> command embedded help documentation
puppetfile	Performs a specified subcommand operation on an environment's Puppetfile. Environment determined by the current working directory. Accepts r10k puppetfile subcommands on page 836.
version	Returns your <code>r10k</code> version.

r10k action options

You can modify `r10k <ACTION>` commands with these options:

Action option	Description
<code>--color</code>	Enable color-coded log messages.
<code>--help</code> or <code>-h</code>	Returns embedded help documentation for a specific action.
<code>--trace</code> or <code>-t</code>	Returns stack traces on application crash.
<code>--verbose</code> or <code>-v</code>	Specify one of the following log verbosity levels: <code>fatal</code> , <code>error</code> , <code>warn</code> , <code>notice</code> , <code>info</code> , <code>debug</code> , <code>debug1</code> , <code>debug2</code>

r10k deploy subcommands

You can use the following subcommand and subcommand options with the `r10k deploy` command.

Subcommand	Description	Subcommand options
display	Returns information about environments and their modules. Refer to: Get environment details with r10k on page 833	<ul style="list-style-type: none"> • <code>-m</code>: Return module information • <code>--detail</code>: Return detailed information • <code>--fetch</code>: Query environment sources so you can check for missing environments • Append one or more environment names to get information about specific environments.
environment	Deploys environment content and modules. Refer to: Updating environments on page 832	<ul style="list-style-type: none"> • Append an environment name to deploy a specific environment. • <code>-m</code>: Install or update modules declared in the Puppetfile • <code>-m --incremental</code>: Install or update only modules set to a "floating" version and any modules whose definitions are new or changed since the last deployment.
module	Deploys modules only. Refer to: Installing and updating modules on page 833	<ul style="list-style-type: none"> • <code>-e</code>: Deploy modules for a specific environment. • Append one or more module names to deploy only specific modules. • <code>no-force</code>: Prevents overwriting local changes to Git-based modules.

Note: The `-p` subcommand option is deprecated. Use `-m` instead.

r10k puppetfile subcommands

The `r10k puppetfile` command accepts these subcommands.

Important: The `puppetfile` command act on the Puppetfile in the current working directory. Make sure you're in the directory containing the Puppetfile for the environment you want to act on.

The `puppetfile` command requires write access to an environment's `modules` directory. Either switch to an appropriately-provisioned user or use elevated privileges to run these commands.

Append subcommands after the `puppetfile` action, such as:

```
r10k puppetfile install
```

Subcommand	Description	Subcommand options
check	Verifies the Puppetfile syntax is correct.	
install	Installs all modules declared in the Puppetfile.	<ul style="list-style-type: none"> • <code>--modules <FILE_PATH></code>: Install modules in the current working directory from a Puppetfile at the supplied file path. • <code>--moduledir <FILE_PATH></code>: Install modules declared in the Puppetfile to a custom module directory location. • <code>--update_force</code>: Install modules declared in the Puppetfile and forcefully overwrite local changes to Git-based modules.
purge	Purges modules not declared in the Puppetfile.	

Note: The `--puppetfile` subcommand option is deprecated. Use `--modules` instead.

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 837

The console certificate expires every 824 days. Regenerate the console certificate when it is nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.

- [Regenerate the SAML certificate](#) on page 838

By default, the SAML certificate expires every 824 days. Regenerate the certificate when it is nearing or past expiration.

- [Regenerate infrastructure certificates](#) on page 838

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 841

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 843

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.

- [Generate a custom Diffie-Hellman parameter file](#) on page 845

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 845

To comply with security regulations, TLSv1 and TLSv1.1 are disabled by default in 2021.7.z versions of Puppet Enterprise (PE).

Regenerate the console certificate

The console certificate expires every 824 days. Regenerate the console certificate when it is nearing or past expiration, or if the certificate is corrupted and you're unable to access the console.

To check the expiry date of your current certificate, run this command on your primary server:

```
/opt/puppetlabs/puppet/bin/openssl x509 -in /etc/puppetlabs/puppet/ssl/
certs/console-cert.pem -noout -startdate -enddate
```

To generate a new console certificate, remove the existing certificate. After you remove the existing certificate, a new one is generated automatically on the next Puppet run.

1. Remove the existing console certificate.

On your primary server, run both these commands:

```
puppet ssl clean --certname console-cert
puppetserver ca clean --certname console-cert
```

2. Run Puppet to generate a new certificate.

On the primary server, run:

```
puppet agent -t
```

Alternatively, you can wait for the next Puppet run.

Regenerate the SAML certificate

By default, the SAML certificate expires every 824 days. Regenerate the certificate when it is nearing or past expiration.

To check the expiry date of your current certificate, run this command on your primary server:

```
/opt/puppetlabs/puppet/bin/openssl x509 -in /etc/puppetlabs/puppet/ssl/certs/saml-cert.pem -noout -startdate -enddate
```

To generate a new SAML certificate, remove the existing certificate. After you remove the existing certificate, a new one is generated automatically on the next Puppet run.

1. Remove the existing SAML certificate.

On the primary server, run both these commands:

```
puppet ssl clean --certname saml-cert
puppetserver ca clean --certname saml-cert
```

2. Run Puppet to generate a new certificate.

On the primary server, run:

```
puppet agent -t
```

Alternatively, you can 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: <ol style="list-style-type: none"> 1. Delete and recreate the certificate authority on page 841 2. Regenerate compiler certificates on page 839, if applicable 3. Regenerate agent certificates on page 840 4. Regenerate replica certificates on page 840
Fix a compromised or damaged certificate authority.	Regenerate compiler certificates on page 839
Fix a compromised compiler certificate or troubleshoot SSL errors on compilers.	Regenerate agent certificates on page 840
Fix a compromised agent certificate or troubleshoot SSL errors on agent nodes.	Regenerate primary server certificates on page 839
Specify a new DNS alt name or other trusted data.	Regenerate primary server certificates on page 839

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.

Regenerate primary server certificates

Regenerate primary server certificates to specify a new DNS alt name or other trusted data. This process regenerates the certificates for all primary infrastructure nodes, including standalone PE-PostgreSQL nodes.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see [Bolt OpenSSH configuration options](#).

To view all available parameters, use the `--help` flag. The logs for all `puppet infrastructure run` Bolt plans are located at `/var/log/puppetlabs/installer/bolt_info.log`.

On your primary server, log in as root, and run:

```
puppet infrastructure run regenerate_primary_certificate
```

You can specify these optional parameters:

- Use `dns_alt_names` to provide a comma-separated list of alternate DNS names to be added to the certificates generated for your primary server.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your primary server certificate.

- Use `--tmpdir` to specify a path to a directory to use for uploading and executing temporary files.

Tip: You might need to set this parameter if [the task fails with a permission denied error](#).

- Use `--force` to force certificate regeneration in situations where your infrastructure is unhealthy due to a damaged certificate.

Regenerate compiler certificates

Regenerate compiler certificates to fix a compromised certificate or troubleshoot SSL errors on compilers, or if you recreated your certificate authority.

On your primary server, log in as root, and run the following command. Specify any additional parameters required for your environment and use case.

```
puppet infrastructure run regenerate_compiler_certificate
target=<COMPILER_HOSTNAME>
```

- **If you use DNS alternative names**, specify `dns_alt_names` as a comma-separated list of names to add to agent certificates.

Important: To ensure naming consistency, if your `puppet.conf` file includes a `dns_alt_names` entry, you must include the `dns_alt_names` parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- **If you recreated your certificate authority**, or are otherwise unable to connect to the compiler with the orchestrator, specify `--use-ssh` and any additional parameters needed to connect over SSH.

The compiler's SSL directory is backed up to `/etc/puppetlabs/puppet/ssl_bak`, its certificate is regenerated and signed, a Puppet run completes, and the compiler resumes its role in your deployment.

Regenerate agent certificates

Regenerate *nix or Windows agent certificates to fix a compromised certificate or troubleshoot SSL errors on agents, or if you recreated your certificate authority.

On your primary server, log in as root (or the administrator), and run the following command. Specify 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 can't connect to nodes with the orchestrator:** Specify clean_crl=true and --use-ssh, as well as any additional parameters needed to connect over SSH.
- **If you want to use a PuppetDB query to generate certificates for multiple agents:** Specify the agent_pdb_query parameter to provide a query to use to collect a list of agents for which you want to regenerate certificates. Make sure the query only returns certnames, such as:

```
facts[certname] { name='domain' and value ~ 'agent.node.com' }
```

If you specify both agent and agent_pdb_query, the query results are combined with the specified agents.

- **If you want to include custom attributes or extension requests in regenerated certificates:** Ensure that agents have the csr_attributes.yaml file containing the necessary custom attributes and extension requests, or specify the optional custom_attributes and extension_requests parameters in the plan. For example:

```
puppet infrastructure run regenerate_agent_certificate
agent=<agent_certname> extension_requests='{"pp_environment":
"development"}'
```

For more information, see [CSR attributes and certificate extensions](#).

Agent SSL directories are backed up to /etc/puppetlabs/puppet/ssl_bak (*nix) or C:/ProgramData/PuppetLabs/puppet/etc/ssl_bak (Windows), their certificates are regenerated and signed, a Puppet run completes, and the agents resume their role in your deployment.

Regenerate replica certificates

Regenerate replica certificates for your disaster recovery installation to specify a new DNS alt name or other trusted data, or if you recreated your certificate authority.

On your primary server, log in as root, and run the following command. Specify any additional parameters required for your environment and use case.

```
puppet infrastructure run regenerate_replica_certificate
target=<REPLICA_HOSTNAME>
```

- **If you use DNS alternative names,** specify dns_alt_names as a comma-separated list of names to add to agent certificates.

Important: To ensure naming consistency, if your puppet.conf file includes a dns_alt_names entry, you must include the dns_alt_names parameter and pass in all alternative names included in the entry when regenerating your agent certificates.

- If you recreated your certificate authority, or are otherwise unable to connect to the replica with the orchestrator, specify `--use-ssh` and any additional parameters needed to connect over SSH.

The replica's SSL directory is backed up to `/etc/puppetlabs/puppet/ssl_bak`, its certificate is regenerated and signed, a Puppet run completes, and the replica resumes its role in your deployment.

Delete and recreate the certificate authority

Recreate the certificate authority only if you're upgrading to the new certificate architecture introduced in Puppet 6.0, or if your certificate authority was compromised or damaged beyond repair.

Before you begin

The `puppet infrastructure run` command leverages built-in Bolt plans to automate certain management tasks. To use this command, you must be able to connect using SSH from your primary server to any nodes that the command modifies. You can establish an SSH connection using key forwarding, a local key file, or by specifying keys in `.ssh/config` on your primary server. For more information, see [Bolt OpenSSH configuration options](#).



CAUTION: Replacing your certificate authority invalidates all existing certificates in your environment. Complete this task only if and when you're prepared to regenerate certificates for both your infrastructure nodes (including external PE-PostgreSQL in extra-large installations) and your entire agent fleet.

On your primary server, log in as root and run:

```
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.

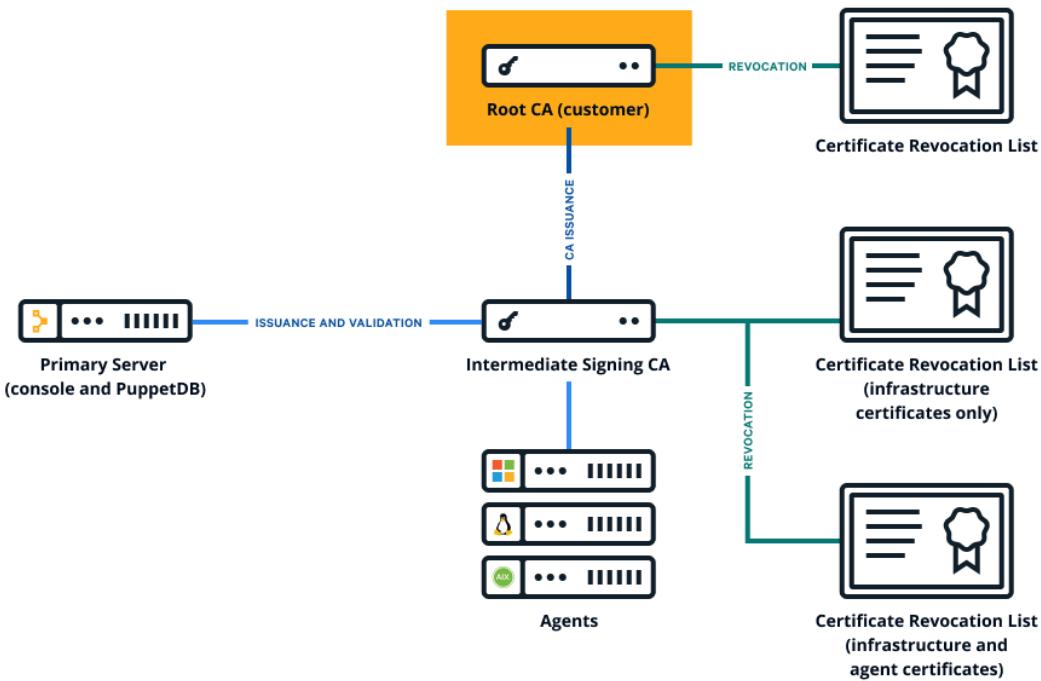
Use an independent intermediate certificate authority

The built-in Puppet certificate authority automatically generates a root and intermediate certificate, but if you need additional intermediate certificates or prefer to use a public authority CA, you can set up an independent intermediate certificate authority. You must complete this configuration during installation.



CAUTION: This method requires more manual maintenance than the default certificate authority setup. With an external chain of trust, you must monitor for and promptly update expired CRLs, because an expired CRL anywhere in the chain causes certificate validation failures. To manage an external CRL chain:

- Take note of the `Next_Update` dates of the CRLs for your entire chain of trust.
- Submit updated CRLs to Puppet Server using the [Certificate Revocation List](#) endpoint.
- Configure agents to download CRL updates by setting `crl_refresh_interval` in the `puppet_enterprise::profile::agent` class.



1. Collect the PEM-encoded certificates and CRLs for your organization's chain of trust, including the root certificate, any intermediate certificates, and the signing certificate. (The signing certificate might be the root or intermediate certificate.)
2. Create a private RSA key, with no passphrase, for the Puppet CA.
3. Create a PEM-encoded Puppet CA certificate.
 - a) Create a CSR for the Puppet CA.
 - b) Generate the Puppet CA certificate by signing the CSR using your external CA.

Ensure the CA constraint is set to true and the `keyIdentifier` is composed of the 160-bit SHA-1 hash of the value of the bit string `subjectPublicKeyfield`. See RFC 5280 section 4.2.1.2 for details.
4. Concatenate all of the certificates into a PEM-encoded certificate bundle, starting with the Puppet CA cert and ending with your root certificate.

```

-----BEGIN CERTIFICATE-----
<PUPPET CA CERTIFICATE>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<OPTIONAL INTERMEDIATE CA CERTIFICATES>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
<ROOT CA CERTIFICATE>
-----END CERTIFICATE-----
  
```

- 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 842), CRL chain (from step 5 on page 843), and private key (from step 2 on page 842) 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
...

```

Related information

[Certificates installed](#) on page 124

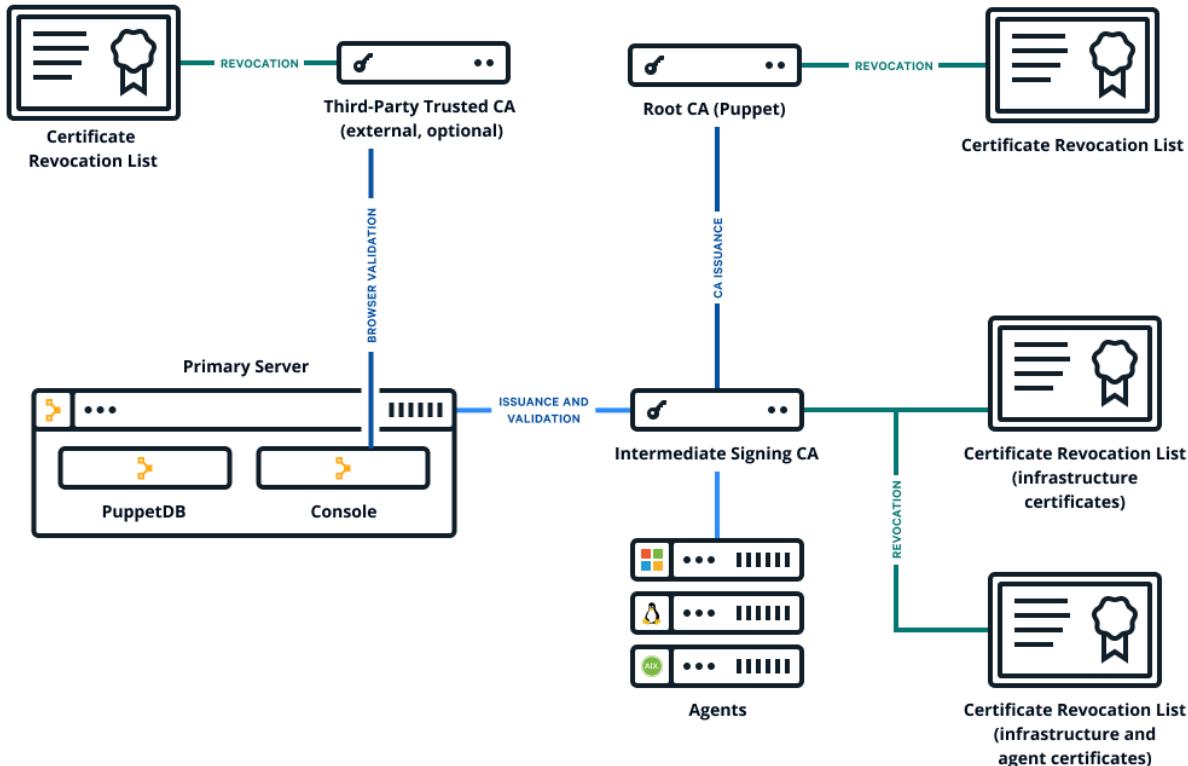
During installation, the software generates and installs a number of SSL certificates so that agents and services can authenticate themselves.

Use a custom SSL certificate for the console

The Puppet Enterprise (PE) console uses a certificate signed by PE's built-in certificate authority (CA). Because this CA is specific to PE, web browsers don't know it or trust it, and you have to add a security exception in order to access the console. If you find that this is not an acceptable scenario, you can use a custom CA to create the console's certificate.

Before you begin

- You need a X.509 cert, signed by the custom party CA, in PEM format, with matching private and public keys.
- If your custom cert is issued by an intermediate CA, the CA bundle must contain a complete chain, including the applicable root CA.
- These keys and certs must be in PEM format.



1. Retrieve the custom certificate and private key.
2. On your primary server, place the certificate and private key into the correct directory locations as follows:
 - Place the certificate in `/etc/puppetlabs/puppet/ssl/certs/console-cert.pem`, replacing any existing file named `console-cert.pem`.
 - Place the private key in `/etc/puppetlabs/puppet/ssl/private_keys/console-cert.pem`, replacing any existing file named `console-cert.pem`.

Important: If your installation includes disaster recovery, repeat this step to place a certificate and a private key configured for your primary server replica in the corresponding directories on the replica.

3. If you previously specified a custom SSL certificate, remove the `browser_ssl_cert` and `browser_ssl_private_key` parameters specified in the console or Hiera data.

In the PE console, go to the **Node groups** page, expand the **PE Infrastructure** group, and select the **PE Console** node group. Then, on the **Classes** tab, locate the `puppet_enterprise::profile::console` class, remove any `browser_ssl_cert` and `browser_ssl_private_key` parameters, and commit changes.

If you declared these parameters on the **Configuration data** tab, remove them from that tab and commit changes.

If you declared these parameters with Hiera, remove them from your Hiera data. For more information, refer to [Configure settings with Hiera](#) on page 212.

4. Run Puppet: `puppet agent -t`

You can navigate to your console and see the custom certificate in your browser.

Related information

[Certificates installed](#) on page 124

During installation, the software generates and installs a number of SSL certificates so that agents and services can authenticate themselves.

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

To comply with security regulations, TLSv1 and TLSv1.1 are disabled by default in 2021.7.z versions of Puppet Enterprise (PE).

You must enable TLSv1 to install agents on these platforms:

- AIX
- Solaris 11



CAUTION: For nodes that use TLSv1, using a script to install or upgrade agents can fail if the curl version installed on the node uses OpenSSL earlier than version 1.0. This issue produces an SSL error during any curl connection to the primary server. As a workaround, add `--ciphers AES256-SHA` to `~/.curlrc` so that curl calls always use an appropriate cipher.

1. In the PE console, navigate to **Node groups > PE Infrastructure**.
2. On the **Configuration data** tab, find or add the `puppet_enterprise::master::puppetserver` class.
3. Add the `ssl_protocols` parameter and set the value to an array of strings representing allowed TLS versions. For example:

```
[ "TLSv1.3", "TLSv1.2", "TLSv1.1", "TLSv1" ]
```

4. Click **Add data** and commit changes.

- Run Puppet on the primary server and any compilers.

Tip: There are several ways to [Run Puppet on demand](#) on page 604.

Related information

[Configure SSL protocols](#) on page 224

You can change what SSL protocols your Puppet Enterprise (PE) infrastructure uses.

Maintenance

- [Back up and restore PE](#) on page 846

Use the Puppet Enterprise (PE) backup tool to create regular backups of your installation. Then, if you migrate your primary server to a new operating system or replace your primary server hardware, you can restore your installation. The backup and restore process can also be useful for troubleshooting or for recovering your installation after a system failure.

- [Database maintenance](#) on page 853

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

- [Rotating the inventory service secret key](#) on page 854

The inventory service uses a randomly-generated secret key to encrypt a connection entry's sensitive parameters.

Back up and restore PE

Use the Puppet Enterprise (PE) backup tool to create regular backups of your installation. Then, if you migrate your primary server to a new operating system or replace your primary server hardware, you can restore your installation. The backup and restore process can also be useful for troubleshooting or for recovering your installation after a system failure.

The PE backup tool is designed for backing up and restoring standard or large PE installations. For extra-large installations, consider using an alternative backup or snapshot process.

If you have a standard or large PE installation, you can implement the following backup and restore process:

- Use the `puppet-backup create` command to back up the primary server.
- When necessary, use the `puppet-backup restore` command to restore the primary server and your PE infrastructure.

Disaster recovery consideration

If your PE installation includes disaster recovery, then after restoring the primary server, you must remove the existing replica and provision a new one.

Important: When upgrading your operating system to a new major version, you must:

- Back up PE on your existing primary server.
- Install a new PE primary server on a node with the upgraded operating system.
- Restore the PE backup on the new primary server.
- For installations with compilers:
 - Remove existing compilers and install new compilers on the upgraded operating system.
- For installations with disaster recovery:
 - Remove the existing replica and provision a new replica on the upgraded operating system.

Restriction: You cannot use the `puppet-backup` command to back up or restore the following components:

- The replica of your primary server

- Compilers
- Secret keys

Customize scope of backup and restore

You can use the `--scope` option to customize what data is backed up or restored.

By default, the `puppet-backup create` command backs up the following data, and the `puppet-backup restore` command restores the same data:

- Your PE configuration, including license, classification, and RBAC settings. However, the configuration backup data 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 the time of the backup.
- PuppetDB data, including facts, catalogs and historical reports.



CAUTION: The `puppet-backup` command does not include secret keys. You must back up this data separately and securely.

If you want to have discrete backup files, or if you want to back up some parts of your infrastructure more often than others, you can use `--scope` command line option to limit the scope of a backup or restore. The `--scope` option accepts one or more of `certs`, `code`, `config`, or `puppetdb`. If unspecified, the default value is `all`. For details about what is and isn't included in each scope, refer to [Directories and data in backups](#) on page 851.

For example, if you have frequent code changes, you might back up your Puppet code more often than you back up the rest of your infrastructure. When you limit the backup scope, the backup file contains only the parts of your infrastructure that you specify. Be sure to identify the scope in your backup file's name so you know what each file contains.

When you restore your primary server, you must restore your Puppet configuration, certificates, code, and PuppetDB data. However, you can restore each aspect from different files, either by using backup files that have limited scopes or by limiting the restore scope. For example, by limiting the scope when you run the `puppet-backup restore` command, you could restore Puppet code, configuration, and certificates from one backup file, and then restore PuppetDB from a different backup file.

Important: When you restore your primary server, you must restore all four data sets: configuration, certificates, code, and PuppetDB. However, you can't restore data that isn't contained in the backup file you're restoring from. For example, a backup file that only contains PuppetDB data can only be used to restore PuppetDB data. In this case you'll need to run the `puppet-backup restore` command multiple times, restoring a different file each time, until you have restored all four data sets (configuration, certificates, code, and PuppetDB).

Back up your infrastructure

The backup process creates a copy of your primary server, including configuration, certificates, code, and PuppetDB. Backup can take several hours depending on the size of PuppetDB.

Before you begin

If you want to encrypt your backup, you must import your GPG public key to your primary server.

To create a complete set of backup data, you need to backup your infrastructure's secret keys and use the `puppet-backup` command to backup your PE configuration, PE certificates, Puppet code, and PuppetDB data. For details about the data included in backup files, refer to [Customize scope of backup and restore](#) on page 847.

1. To ensure that `pg_repack` doesn't run during the backup process, stop the `pe_databases` module timers:

```
systemctl stop pe_databases-*.timer
```

- Run the `puppet-backup` command on your primary server. The default command is:

```
sudo puppet-backup create --dir=<BACKUP_DIRECTORY>
```

You can customize your backup by specifying the following optional parameters:

- `--dir=<BACKUP_DIRECTORY>`: Specify a separate a secure location for your backup.
- `--name=<BACKUP_NAME>`: Specify the backup file's name. The default name is `pe_backup` with a timestamp indicating when the backup file was created, such as: `pe_backup-<TIMESTAMP>.tgz`.
- `--pe-environment=<ENVIRONMENT>`: Specify an environment to back up. To ensure the configuration is recovered correctly, this must be the environment where your primary server is located. The default value is `production`.
- `--scope=<SCOPE_LIST>`: Specify the data you want the backup file to contain. This is used for [Customize scope of backup and restore](#) on page 847. The default scope is `all`. To backup specific data, limit the scope by using one or more of the following values: `certs`, `code`, `config`, or `puppetdb`.

Tip: If you specify `--scope`, specify a `--name` that describes the file's scope.

- `--gpgkey=<KEY_ID>`: Specify a GPG key ID to use to encrypt the backup file.
- `--force`: Specify this parameter if you want to bypass validation checks and ignore warnings.

- Back up the secret keys directory and, if applicable, the secret key for the LDAP service.



CAUTION:

The `puppet-backup` command does not include secret keys. You must back up this data separately.
Secure the keys as you would any sensitive information.

- The secret keys directory is located at: `/etc/puppetlabs/orchestration-services/conf.d/secrets/`
- (If applicable) The LDAP service key is located at: `/etc/puppetlabs/console-services/conf.d/secrets/keys.json`

- Restart the `pe_databases` module timers:

```
systemctl start pe_databases-catalogs.timer pe_databases-facts.timer  
pe_databases-other.timer
```

Each time you use `puppet-backup` to create a new backup, PE creates a single backup file containing everything you're backing up (defined by the `--scope`). PE writes backup files to `/var/puppetlabs/backups`, unless you specify a different location in the `puppet-backup` command. The file name follows the default naming convention (`pe_backup-<TIMESTAMP>.tgz`), unless you specified a different name in the `puppet-backup` command.

Restore your infrastructure

Use the restore process when you migrate your primary server to a new operating system or to a new host. You can also use the restore process to recover your installation after a system failure.

Before you begin

You must have created backup files, as described in [Back up your infrastructure](#) on page 847.

You must import the GPG key pair (both the public and private keys) that you used for encryption to your new primary server. The GPG keys are required to decrypt an encrypted backup.

- To ensure that the `pg_repack` extension doesn't run, stop the `pe_databases` module timers:

```
systemctl stop pe_databases-*.timer
```

2. If you are restoring the primary server to a node that is currently hosting an active primary server, you must first purge the existing installation and reinstall PE. To do this:
 - a) On the primary server, run this command to uninstall PE:

```
sudo /opt/puppetlabs/bin/puppet-enterprise-uninstaller -p -d
```

- b) Ensure that these directories are no longer present on the system: `/opt/puppetlabs/` and `/etc/puppetlabs/`

3. Install PE on the primary server you want to restore to. You must install the same PE version used to create your backup files.
 - a) If you don't have the PE installer script on the machine you want to restore to, [download](#) the installer tarball and unpack it by running: `tar -xf <TARBALL_FILENAME>`
 - b) Navigate to the directory containing the install script. Normally, the installer script is located in the PE directory that is created when you unpacked the tarball.
 - c) To install PE, run: `sudo ./puppet-enterprise-installer`
4. On your primary server, run the `puppet-backup restore` command to restore your PE infrastructure. The default command is:

```
sudo puppet-backup restore <BACKUP-Filename>
```

You must use the following parameter to specify the backup file you want to restore from:

- `<BACKUP-Filename>`: The location, name, and contents of your backup files are determined when you [Back up your infrastructure](#) on page 847.

You can customize your restore by specifying the following optional parameters:

- `--pe-environment=<ENVIRONMENT>`: Specify an environment to restore. The default value is `production`.
- `--scope=<SCOPE_LIST>`: Specify the data you want to restore. This is used for [Customize scope of backup and restore](#) on page 847. The default scope is `all`. If you only want to restore specific data, specify one or more of: `certs`, `code`, `config`, or `puppetdb`.
- `--force`: Specify this parameter if you want to bypass validation checks and ignore warnings.

Important: You might have to run the `puppet-backup restore` command again if either the backup file or your `restore` command had limited `--scope`. For more information, refer to [Customize scope of backup and restore](#) on page 847.

5. Restore your secret key backups. These keys are used to encrypt and decrypt sensitive data in the inventory service, orchestrator, and the LDAP service (if enabled).
 - The location of the Inventory service and orchestrator keys directory is: `/etc/puppetlabs/orchestration-services/conf.d/secrets/`
 - The location of the LDAP service key file is: `/etc/puppetlabs/console-services/conf.d/secrets/keys.json`



CAUTION: The `puppet-backup restore` command does not include secret keys. You must restore this data separately.

6. Make sure the inventory service's secret key ownership is configured as: `pe-orchestration-services:pe-orchestration-services`
7. If the LDAP service is enabled, make sure the LDAP service's secret key ownership is configured as: `pe-console-services:pe-console-services`
8. Restart `pe-orchestration-services` and `pe-console-services`.

9. 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, you need to redirect your agents to the new primary server. One way to do this is by running a task with the Bolt task runner:
 - a) If Bolt isn't already installed, download and [install](#) Bolt.
 - b) To update the `puppet.conf` file to point all agents at the new primary server, run these commands:

```
bolt task run puppet_conf action=set section=agent setting=server
  value=<RESTORE_HOSTNAME> --targets <COMMA-SEPARATED_LIST_OF_NODES>
bolt task run puppet_conf action=set section=main setting=server
  value=<RESTORE_HOSTNAME> --targets <COMMA-SEPARATED_LIST_OF_NODES>
```

- c) On the newly restored primary server, run `puppet agent -t --no-use_cached_catalog` to apply the changes. Then run the same command again to restart services.

Important: You must run the command twice.

- d) To test the connection to the new primary server, on each agent node, run the following command:

```
puppet agent -t --no-use_cached_catalog
```

10. If Code Manager was enabled when you created your backup file, complete the following steps on the newly restored primary server:

- a) Run the following command:

```
puppet-access login
```

- b) For each code environment you want to deploy, run:

```
puppet code deploy --wait <ENVIRONMENT_NAME>
```

Alternatively, if you want to deploy all of your PE code environments, you can run the following command:

```
puppet code deploy --wait --all
```

- c) Run the following:

```
puppet agent -t --no-use_cached_catalog
puppet agent -t --no-use_cached_catalog
```

Important: You must run the command twice.

11. If your installation includes compilers and you are restoring your primary server on the same operating system that was in use when the backup was created, run the following command on all compilers:

```
puppet agent -t --server_list <PRIMARY_CERTNAME>
```

12. If your installation includes compilers and you are restoring your primary server after changing or upgrading your operating system, you must complete the following steps:

- a) On the primary server, run the following command for each compiler:

```
puppet node purge <COMPILER_CERTNAME>
```

- b) If any of the compilers included in your installation were legacy compilers (compilers without the PuppetDB service), you must unpin them from the **PE Master** node group.
- c) Install and configure new compilers.

Important: Compilers must run the same OS major version, platform, and architecture as the primary server.

13. If your installation includes a primary server replica, you must complete the following steps:

- On the primary server, run the following command:

```
puppet infrastructure forget <REPLICA_CERTNAME>
```

- Provision and enable a new replica.

Related information

[Uninstalling](#) on page 176

Puppet Enterprise (PE) includes a script for uninstalling. You can uninstall infrastructure nodes or uninstall the agent from agent nodes.

[Installing PE](#) on page 124

To install Puppet Enterprise (PE), you can use either the PE installer tarball for your operating system platform or Puppet Installation Manager.

[Provision and enable a replica](#) on page 259

Provisioning a replica duplicates specific components and services from the primary server to the replica. Enabling a replica activates most of its duplicated services and components, and instructs agents and infrastructure nodes how to communicate in a failover scenario.

Directories and data in backups

These directories and data are included in PE backups.

A default `puppet -backup` command captures all scopes, meaning all directories and data described in the table below. However, you can use the `--scope` option to limit the contents of backup files, or to restore data from multiple backup files, as described in [Customize scope of backup and restore](#) on page 847. In this case, the `--scope` option indicates which directories and data to back up or restore.

Scope	Directories and databases
<code>certs</code> (PE certificates)	<ul style="list-style-type: none"> PE CA certificates <code>/etc/puppetlabs/puppet/ssl/</code>
<code>code</code> (Puppet code)	<p>This scope captures the Puppet code deployed to your code directory at the time of the backup. Specifically:</p> <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> <code>/opt/puppetlabs/server/data/orchestration-services/data-dir</code> <code>/opt/puppetlabs/server/data/orchestration-services/code)</code>

Scope	Directories and databases
config (PE configuration)	<p>This scope captures your PE configuration, including license, classification, and RBAC settings. Some directories and data are excluded, such as Puppet gems, Puppet Server gems, and directories captured in other scopes. Specifically, it includes:</p> <ul style="list-style-type: none"> • The orchestrator, RBAC, and classifier databases • The contents of <code>/etc/puppetlabs/</code>, except: <ul style="list-style-type: none"> • The <code>/code</code> and <code>/code-staging</code> directories, which are included in the code scope. • The <code>/puppet/ssl</code> directory, which is included in the certs scope. • The contents of <code>/opt/puppetlabs/</code>, except: <ul style="list-style-type: none"> • <code>/puppet</code> • <code>/server/pe_build</code> • <code>/server/data/packages</code> • <code>/server/apps</code> • <code>/server/data/postgresql</code> • <code>/server/data/enterprise/modules</code> • <code>/server/data/puppetserver/vendored-jruby-gems</code> • <code>/bin</code> • <code>/client-tools</code> • <code>/server/share</code> • <code>/server/data/puppetserver/filesync/storage</code> • <code>/server/data/puppetserver/filesync/client</code> • <code>/server/data/orchestration-services/data-dir</code> and <code>/server/data/orchestration-services/code</code>, which are included in the code scope.
puppetdb (PuppetDB)	<ul style="list-style-type: none"> • PuppetDB data, including facts, catalogs, and historical reports • The contents of <code>/opt/puppetlabs/server/data/puppetdb</code>, except the <code>/stockpile</code> directory.

Database maintenance

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

Enable the `pe_databases` module

The `pe_databases` module helps you manage and tune your Puppet Enterprise (PE) databases. The module is installed in the `$basemodulepath` directory as part of the PE installation or upgrade process, and it is enabled by default.

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

1. To enable or disable the `pe_databases` module, change the `puppet_enterprise::enable_database_maintenance` parameter. This parameter accepts Boolean values.
2. Run Puppet: `puppet agent -t`

Related information

[How to configure PE](#) on page 211

After you've installed Puppet Enterprise (PE), you can optimize it by configuring and tuning settings. 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.

Databases in PE

Puppet Enterprise (PE) uses PostgreSQL as the backend for its databases. You can use the native tools in PostgreSQL to perform database exports and imports.

The PE PostgreSQL database includes the following databases:

Database	Description
<code>pe-activity</code>	Activity data from the Classifier, including users, nodes, and times of activities
<code>pe-classifier</code>	Classification data, all node group information
<code>pe-puppetdb</code>	PuppetDB data, including exported resources, catalogs, facts, and reports
<code>pe-rbac</code>	Role-based access control (RBAC) data, including users, permissions, and AD/LDAP information
<code>pe-orchestrator</code>	Orchestrator data, including user, node, and job run result details

List all database names

You can generate a list of PostgreSQL database names.

1. Switch to the `pe-postgres` user by running:

```
sudo su - pe-postgres -s /bin/bash
```

2. Open the PostgreSQL command-line by running:

```
/opt/puppetlabs/server/bin/psql
```

3. To list the databases, run: `\l`
4. To exit the PostgreSQL command line, run: `\q`

- To log out of the `pe-postgres` user, run: `logout`

Rotating the inventory service secret key

The inventory service uses a randomly-generated secret key to encrypt a connection entry's sensitive parameters.

Rotate the inventory service secret key

Rotate the secret key every 90 days to reduce the probability of an attacker compromising the secret key.

- Stop the inventory service on the primary server. You can use the command `puppet resource service pe-orchestration-services ensure=stopped`, where the `pe-orchestration-services` service contains both the orchestrator and inventory services.
- Stop the Puppet service to ensure that a periodic Puppet run does not accidentally start the inventory service while you are rotating the secret key.
- Use this command to download the `key_rotation.rb` script:

```
curl https://puppet.com/docs/pe/latest/files/key_rotation.rb -L --output key_rotation.rb
```

- Run the `key_rotation.rb` script on the primary server. You must log in as root or use `sudo` to run the script with [elevated privileges](#).

The `key_rotation.rb` script:

- Calculates the secret key directory and database URL by reading the inventory service's config file.
- Generates the new key and writes it to `<SECRET_KEY_DIR>/new_key.json`.
- Uses `psql` to re-encrypt the old data with the new key.
- Moves the new key to the old key's location (`<SECRET_KEY_DIR>/keys.json`).

If the inventory service's database is on a different host than the primary server, you must specify the URL using the `DATABASE_URL` environment variable. This must be a valid PostgreSQL URL. For example, the following invocation connects to the `inventory_service` database as the `inventory_user` with the password `inventory_password` on host `remote_db_host`:

```
DATABASE_URL=postgres://inventory_user:inventory_password@remote_db_host/inventory_service key_rotation.rb
```

If re-encryption fails, you can re-run the script. The script does not generate another new key; instead, it detects the previously-created new key and skips to reattempt re-encryption.

If moving the new key to old key's location fails, you must manually move the new key to the old key's location. To do this, you can run:

```
mv <SECRET_KEY_DIR>/new_key.json <SECRET_KEY_DIR>/keys.json
```

For example:

```
mv etc/puppetlabs/orchestration-services/conf.d/secrets/new_key.json etc/puppetlabs/orchestration-services/conf.d/secrets/keys.json
```

- Delete the `key_rotation.rb` script to prevent unintentional secret key rotations.
- Restart the inventory service on the primary server by running: `puppet resource service pe-orchestration-services ensure=running`
- Restart the Puppet service.

[Back up your infrastructure](#) to capture the new secret key and re-encrypted data.

Troubleshooting

Use this guide to troubleshoot issues with your Puppet Enterprise (PE) installation.

Important: Before following troubleshooting guidance, review [What gets installed and where](#). PE installs several software components, configuration files, databases, logs, services, and users. It is useful to know their locations when you need to troubleshoot your infrastructure.

- [Log locations](#) on page 855

The software distributed with Puppet Enterprise (PE) generates log files you can use for troubleshooting.

- [Troubleshooting installation](#) on page 858

If installation fails, check for these issues.

- [Troubleshooting disaster recovery](#) on page 859

If disaster recovery commands fail, check for these issues.

- [Troubleshooting puppet infrastructure run commands](#) on page 859

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 860

If agent nodes can't retrieve configurations, check for communication, certificate, DNS , and NTP issues.

- [Troubleshooting the databases](#) on page 862

Use these strategies to troubleshoot issues with the databases that support the console.

- [Troubleshooting cloud deployments](#) on page 863

If you encounter issues with a PE cloud deployment, review the troubleshooting information.

- [Troubleshooting SAML connections](#) on page 864

There are some common issues and errors that can occur when connecting a SAML identity provider to PE, such as failed redirects, rejected communications, and failed group binding.

- [Troubleshooting backup and restore](#) on page 865

If backup or restore fails, check for these issues.

- [Troubleshooting Code Manager](#)

- [Troubleshooting Windows](#) on page 866

Troubleshoot issues in Windows PE installations, such as failed installations, failed upgrades, problems applying manifests, and other issues.

Log locations

The software distributed with Puppet Enterprise (PE) generates log files you can use for troubleshooting.

Primary server logs

Code Manager access log

Location: `/var/log/puppetlabs/puppetserver/code-manager-access.log`

File sync access log

Location: `/var/log/puppetlabs/puppetserver/file-sync-access.log`

Puppet Communications Protocol (PCP) broker log

This is the log file for PCP brokers on compilers.

Location: `/var/log/puppetlabs/puppetserver/pcp-broker.log`

General Puppet Server log

This is where the primary server logs its activity, including compilation errors and deprecation warnings.

Location: `/var/log/puppetlabs/puppetserver/puppetserver.log`

Puppet Server access log

Location: /var/log/puppetlabs/puppetserver/puppetserver-access.log

Puppet Server daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/puppetserver/puppetserver-daemon.log

Puppet Server status log

Location: /var/log/puppetlabs/puppetserver/puppetserver-status.log

Agent logs

The agent log locations depend on the agent node's operating system.

On *nix nodes, the agent service logs activity to the syslog service. The node's operating system and syslog configuration determines where these messages are saved. The default locations are as follows:

- Linux: /var/log/messages
- macOS: /var/log/system.log
- Solaris: /var/adm/messages

On Windows nodes, the agent service logs its activity to the Event Log. Browse the Event Viewer to view those messages. You might need to enable [Logging and debugging](#) on page 870.

Console and console services logs

General console services log

Location: /var/log/puppetlabs/console-services/console-services.log

Console services API access log

Location: /var/log/puppetlabs/console-services/console-services-api-access.log

Console services access log

Location: /var/log/puppetlabs/console-services-access.log

Console services daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/console-services-daemon.log

NGINX access log

Location: /var/log/puppetlabs/nginx/access.log

NGINX error log

Contains console errors that aren't logged elsewhere and errors related to NGINX.

Location: /var/log/puppetlabs/nginx/error.log

Installer logs

HTTP log

Contains web requests sent to the installer.

Only exists on machines from which a web-based installation was performed.

Location: /var/log/puppetlabs/installer/http.log

Orchestrator info log

Contains run details about `puppet infrastructure` commands that use the orchestrator. This includes commands to provision and upgrade compilers, convert legacy compilers, and regenerate agent and compiler certificates.

Location: /var/log/puppetlabs/installer/orchestrator_info.log

Last installer run logs, by hostname

Contains the contents of the last installer run.

There can be multiple log files, labeled by hostname.

Location: /var/log/puppetlabs/installer/install_log.lastrun.<HOSTNAME>.log

Installer operation logs, by timestamp

Captures operations performed during installation and any errors that occurred.

There can be multiple log files, labeled by timestamp.

/var/log/puppetlabs/installer/installer-<TIMESTAMP>.log

Disaster recovery command logs, by action, timestamp, and description

Contains details about disaster recovery command execution.

There can be multiple log files for each command because each action triggers multiple Puppet runs (Some on the primary server and some on the replica).

Location: /var/log/puppetlabs/installer/<ACTION-NAME>_<TIMESTAMP>_<RUN-DESCRIPTION>.log

Bolt info log

Can be valuable when [Troubleshooting disaster recovery](#) on page 859.

Location: /var/log/puppetlabs/installer/bolt_info.log

Database logs

Database logs include PostgreSQL and PuppetDB logs.

PostgreSQL startup log

Can be valuable when [Troubleshooting the databases](#) on page 862.

Location: /var/log/puppetlabs/postgresql/14/pgstartup.log

PostgreSQL daily logs, by weekday

There is one log file for each day of the week. Log file names use short names, such as Mon for Monday, Tue for Tuesday, and so on.

Location: /var/log/puppetlabs/postgresql/14/postgresql-<WEEKDAY>.log

General PuppetDB log

Location: /var/log/puppetlabs/puppetdb/puppetdb.log

PuppetDB access log

Location: /var/log/puppetlabs/puppetdb/puppetdb-access.log

PuppetDB status log

Location: /var/log/puppetlabs/puppetdb/puppetdb-status.log

Orchestration logs

Orchestration logs include orchestration services and related components, such as PXP agent and Bolt server.

Aggregate node count log

Location: /var/log/puppetlabs/orchestration-services/aggregate-node-count.log

Puppet Communications Protocol (PCP) broker log

This is the log file for PCP brokers on the primary server.

Location: /var/log/puppetlabs/orchestration-services/pcp-broker.log

Puppet Communications Protocol (PCP) broker access log

Location: /var/log/puppetlabs/orchestration-services/pcp-broker-access.log

Orchestration services access log

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-access.log

Orchestration services daemon log

This is where you can find fatal errors and crash reports.

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-daemon.log

Orchestration services status log

Location: /var/log/puppetlabs/orchestration-services/orchestration-services-status.log

Puppet Execution Protocol (PXP) agent log

*nix location: /var/log/puppetlabs/pxp-agent/pxp-agent.log

Windows location: C:/ProgramData/PuppetLabs/pxp-agent/var/log/pxp-agent.log

Bolt server log

Can be valuable when [Troubleshooting connections between components](#) on page 860.

Location: /var/log/puppetlabs/bolt-server/bolt-server.log

Node inventory service log

Location: /var/log/puppetlabs/orchestration-services/orchestration-services.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.

Misconfigured DNS

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.

Misconfigured security settings

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 through the IP address that its valid DNS names resolve to, not just through 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 into 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 misconfigured, Bolt tries to connect as root. To make sure the correct user connects to the nodes, you have the following options:

- Set up your OpenSSH `ssh_config` configuration file to point to a user with `sudo` privileges. For example:

```
Host *.example.net
  UserKnownHostsFile=~/ssh/known_hosts
  User <USER_WITH_SUDO_PRIVILEGES>
```

- When running a `puppet infrastructure run` command, include the `--user <USER_WITH_SUDO_PRIVILEGES>` flag.

If your `sudo` configuration requires a password to run commands, include the `--sudo-password <PASSWORD>` flag when running a `puppet infrastructure run` command.

Tip: To avoid logging the 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 the `/etc/sudoers` file, you must do one of the following:

- Remove this option from the file.
- Include 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 wrapped in quotes, 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 a Name or service not known error.

1. Verify that the primary server is reachable at a DNS name your agents recognize.
If you aren't sure how to do this, refer to: [Agents aren't using the primary server's valid DNS name](#) on page 861
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 contain warnings about unverified peer certificates in the current SSL session, the agent's certificate signing request (CSR) that hasn't yet been signed.

1. On the primary server, run `puppet cert list` to generate a list of pending CSRs.
2. **Tip:** You can also [Manage CSRs in the console](#).
2. To sign a node's certificate, run: `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 agent node, if you don't get one of the primary server's valid DNS names (which you chose when installing the primary server) when you run `puppet agent --configprint server`, then the agent node and primary server can't communicate.

1. To edit the primary server's hostname on agent nodes, open the `/etc/puppetlabs/puppet/puppet.conf` file, and change the `server` setting to a valid DNS name.
2. To reset the primary server's valid DNS names, log in as root (or the Administrator) and run:

```
puppet infrastructure run regenerate_primary_certificate --dns_alt_names=<COMMA-SEPARATED_LIST_OF_DNS_NAMES>
```

Time is out of sync

The date and time must be in sync on the primary server and agent nodes.

If time is out of sync on nodes, running the `date` command returns incorrect or inconsistent dates.

Set up NTP to get the time in sync. However, 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, you get invalid dates (such as certificates with future dates) when you run:

```
openssl x509 -text -noout -in $(puppet config print --section master ssldir)/certs/<NODE_NAME>.pem
```

1. On the primary server, delete certificates with invalid dates by running:

```
puppetserver ca clean --certname <NODE_CERT_NAME>
```

2. On the nodes with invalid certificates, delete the SSL directory by running:

```
rm -r $(puppet config print --section master ssldir)
```

3. On each impacted agent node, run `puppet agent --test` to generate a new certificate request.
4. On the primary server, run `puppetserver ca sign <NODE_NAME>` to sign each request.

A node is re-using a certname

If a new node re-uses an old node's certname, and the primary server retains the previous node's certificate, the new node can't request a new certificate.

1. On the primary server, clear the node's certificate by running:

```
puppetserver ca clean --certname <NODE_CERT_NAME>
```

2. On the agent node, run `puppet agent --test` to generate a new certificate.

3. On the primary server, run `puppetserver ca sign <NODE_NAME>` to sign the request.

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 logs contain errors like `could not back up`, this means 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 the filebucket server attribute points to the correct hostname. For example:

```
# 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 the PE Bolt server

There are two options for debugging a faulty connection between the orchestrator and the PE Bolt server.

- Set the `bolt_server_loglevel` parameter in the `puppet_enterprise::profile::bolt_server` class, and then run Puppet.
- Manually update the `loglevel` parameter in the `/etc/puppetlabs/bolt-server/conf.d/bolt-server.conf` file.

The Bolt server logs are located at: `/var/log/puppetlabs/bolt-server/bolt-server.log`

Troubleshooting the databases

Use these strategies to troubleshoot issues with the databases that support the console.

Common issues include:

- [The PostgreSQL database takes up too much space](#) on page 862
- [PostgreSQL buffer memory causes installation to fail](#) on page 862
- Port conflicts, such as: [The PuppetDB default port conflicts with another service](#) on page 863
- Incorrect `puppet apply` configuration, for example: [puppet resource generates Ruby errors after connecting puppet apply to PuppetDB](#) on page 863.
- In unmanaged PostgreSQL installations, you are not on the latest supported version of PostgreSQL. For upgrade instructions, refer to [Upgrade an unmanaged PostgreSQL installation](#) on page 187.

The PostgreSQL database takes up too much space

The PostgreSQL `autovacuum=on` setting prevents the database from growing too large and unwieldy. Routine vacuuming is enabled 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 try to use more shared buffer memory than is available.

If this issue is present, the `pgstartup.log` (located at `/var/log/pe-postgresql/pgstartup.log`) contains the following 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 `shmmmax` kernel setting to approximately 50% of the total RAM.
2. To get the value for the `shmmax` kernel setting, divide the value of the `shmmmax` setting by the page size. To confirm the page size, run: `getconf PAGE_SIZE`

3. Set the new kernel settings by running:

```
sysctl -w kernel.shmmax=<your shmmax calculation>
sysctl -w kernel.shmall=<your shmall calculation>
```

The PuppetDB default port conflicts with another service

By default, PuppetDB communicates over port 8081. In some cases, this might conflict with other services, such as McAfee ePolicy Orchestrator.

Install PuppetDB in text mode with the non-default port specified on the `puppet_enterprise::puppetdb_port` parameter in the `pe.conf` file.

After installation, make sure the `puppetdb_port` value is correct on the [PE Infrastructure node group](#) on page 456.

puppet resource generates Ruby errors after connecting puppet apply to PuppetDB

If `puppet apply` is configured incorrectly, then `puppet resource` ceases to function and returns a Ruby run error.

An example of an incorrect `puppet apply` configuration would be putting the `storeconfigs_backend = puppetdb` and `storeconfigs = true` parameters in both the `main` and `primary` server sections of the `puppet.conf` file.

You need to modify the `routes.yaml` file (located at `/etc/puppetlabs/puppet/routes.yaml`) so that it correctly [connects puppet apply](#) without impacting other functions.

Troubleshooting cloud deployments

If you encounter issues with a PE cloud deployment, review the troubleshooting information.

- [Authentication fails with SSH username or credentials](#) on page 863
Cloud providers vary in their support for Secure Shell (SSH) authentication.
- [After 60 days, the puppetadmin user account stops working](#) on page 864

The default password of the `puppetadmin` user expires 60 days after the image is created. If you fail to reset the password, the account expires.

- [Agent run fails for non-root users](#) on page 864

An agent run initiated by `puppetadmin` or any other non-root user fails when attempting to access certificates, packages, and services.

- [Certificate-signing curl command has incorrect URL](#) on page 864

The `curl` command on the console's **Unsigned Certificates** page contains a URL that uses your primary server's private hostname or internal DNS name. This type of URL causes issues if nodes cannot resolve the private hostname or internal DNS name.

Authentication fails with SSH username or credentials

Cloud providers vary in their support for Secure Shell (SSH) authentication.

Amazon Web Services (AWS)

AWS cloud deployments use the [cloud-init](#) method to provision an SSH key for the `puppetadmin` user but disable root SSH access. You must specify a key pair when launching an AWS Marketplace image and connect it with the matching private key. For example, you can run:

```
aws ec2 run-instance --key-name <KEYPAIR_NAME> ...
...
ssh -i ~/.ssh/<KEYPAIR_PRIVATE>.pem puppetadmin@<PRIMARY_HOSTNAME>
```

Microsoft Azure

Azure cloud deployments support authentication with either an SSH key pair or a username and password. The private key must be the pair of the public key specified when you created the image.

After 60 days, the `puppetadmin` user account stops working

The default password of the `puppetadmin` user expires 60 days after the image is created. If you fail to reset the password, the account expires.

To prevent the password from expiring, run `chage -E -1 puppetadmin` on the primary server.

Agent run fails for non-root users

An agent run initiated by `puppetadmin` or any other non-root user fails when attempting to access certificates, packages, and services.

Always execute agent runs with super-user privileges:

```
sudo /usr/local/bin/puppet agent -t
```

Certificate-signing curl command has incorrect URL

The `curl` command on the console's **Unsigned Certificates** page contains a URL that uses your primary server's private hostname or internal DNS name. This type of URL causes issues if nodes cannot resolve the private hostname or internal DNS name.

For example, the `curl` command might be similar to the following example:

```
curl -k https://puppetmasterv2.liweiionmsdnwoe.xx.internal.cloudapp.net:8140/packages/current/install.bash | sudo bash
```

To resolve the issue, change the private hostname or internal DNS name to the public hostname or externally qualified domain name.

Troubleshooting SAML connections

There are some common issues and errors that can occur when connecting a SAML identity provider to PE, such as failed redirects, rejected communications, and failed group binding.

Tip: In the case of any SAML connection errors, check the SAML configurations in both PE and your identity provider.

Failed redirects

Redirects fail (with a 404 error code) when there are mismatched URLs between PE and the identity provider. Depending on where the redirect occurs, there are two possible ways to fix this:

- If the redirect fails when going from the identity provider to PE, fix the mismatched URLs in your identity provider's SAML configuration.
- If the redirect fails when going from PE to the identity provider, fix the mismatched URLs in your PE SAML configuration.

Rejected communication requests

If PE or the identity provider rejects communications or returns an error, check the `console-services.log` file (located at `/var/log/puppetlabs/console-services/console-services.log`) for details about the communication failure.

Usually, this means there are mismatched certificates for PE and the identity provider, and that you need to reconfigure the certificates.

Failed user-group binding

If users aren't binding to their assigned groups, or if user permissions are missing, make sure:

- There isn't a mismatch in attribute bindings. Check the attribute binding values in your identity provider and PE SAML configurations.
- Tip:** If unknown attributes appear in output logs at the debug level, this can be an indication of mismatched attribute bindings.
- The group export is incorrect in your identity provider's configuration.

Related information

[Connect to a SAML identity provider](#) on page 290

Use the console to set up SSO or MFA with your SAML identity provider.

SAML error messages

These are common PE error messages related to SAML and how you can troubleshoot them.

Expected login bindings <BINDING> in attributes and it wasn't present.

The identity provider didn't provide a specified login attribute for the user.

Check your identity provider configuration.

Multiple login bindings found in attributes and only one expected.

The identity provider supplied multiple login entries in the assertion but only one entry is allowed.

Check your identity provider configuration.

User '{0}' has been revoked and is unable to login

Either an administrator manually revoked the user's account in PE or RBAC automatically revoked the user's account.

RBAC usually automatically revokes users when the user has no recent activity. This is based on the `account_expiry_days` parameter. For more information, refer to [Configure RBAC and token-based authentication settings](#) on page 224.

If the account was manually revoked, contact the administrator who revoked the account.

SAML library errors

There are various SAML library errors, which are identified by their namespace.

Sometimes these errors are recorded in the `console-services.log` file.

These errors usually indicate a malformed payload, mismatched `entity-id`, or an untrusted certificate.

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 missing or has a different version on the primary server's agent environment, you get this error: `command puppet infrastructure recover_configuration failed`.

To fix this, install the 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 which gems, and which versions, you have installed on your Puppet Server, run: `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, if the restore process exits with errors about a duplicate operator family, follow these steps:

1. Log into your PostgreSQL instance by running:

```
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 issues in Windows PE installations, such as failed installations, failed upgrades, problems applying manifests, and other issues.

If you are experiencing failures when installing or upgrading agents, refer to [Installation fails](#) on page 866 and [Upgrade fails](#) on page 867.

If manifests are failing to be applied, or are being applied incorrectly, refer to [Errors when applying a manifest or doing a Puppet agent run](#) on page 867.

For other issues, refer to the [Error messages](#) on page 868 reference. You might need to enable temporary [Logging and debugging](#) on page 870.

Installation fails

Check for these issues if Puppet agent installation fails on a Windows node.

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
- 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

Installing the Puppet agent requires [elevated privileges](#), such as being logged in as the Administrator or running commands in an Administrator command prompt or PowerShell window.

The following are indications that you are attempting to install without admin privileges:

- Agent installation fails when trying to perform an unattended installation from the command line.
- You get a `norestart` message.
- The installation logs indicate that installation is forbidden by system policy.

Upgrade fails

The Puppet agent `.msi` package overwrites existing entries in the `puppet.conf` file. If you upgrade or reinstall the agent with a different primary server hostname, Puppet applies the new value in `$confdir\puppet.conf` file.

When you upgrade a Windows agent, you must use the same primary server hostname that you specified when you originally [installed the agent](#).

For information on configuring `puppet.conf` and which settings are preserved during upgrades, refer to [MSI properties](#) on page 156.

Errors when applying a manifest or doing a Puppet agent run

If your manifests aren't applied, or are applied incorrectly, on Windows nodes, check for these issues.

Path or file separators are incorrect

For Windows nodes, path separators must use a semi-colon (`;`).

File separators must use forward slashes or backslashes, depending on the attribute. In most resource attributes, the Puppet language accepts either forward slashes or backslashes as the file separator. However, some attributes absolutely require forward slashes, and some attributes absolutely require backslashes.

You must escape backslashes that are double-quoted(`"`). When single-quoted (`'`), escaping is optional. For example, these are all valid file resources:

```
file { 'c:\path\to\file.txt': }
file { 'c:\\path\\to\\\\file.txt': }
file { "c:\\path\\to\\\\file.txt": }
```

However `file { "c:\\path\\to\\file.txt": }` is an invalid path, because `\p`, `\t`, and `\f` are interpreted as escape sequences.

For more information:

- Learn about [Files and paths in the Puppet language on Windows](#) in the Puppet documentation.
- Learn about Windows modifications when [Using example commands](#) on page 28 that you find in the PE documentation.

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 the dependencies correctly. For example, Puppet fails to apply the following manifest because it 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.

To troubleshoot this, use `cmd.exe` to wrap the built-in:

```
exec { 'cmd.exe /c echo foo':
  path => 'c:\windows\system32;c:\windows'
}
```

Tip: In the 32-bit versions of the Puppet agent, you might encounter file system redirection, where `system32` is automatically switched to `sysWoW64`.

PowerShell scripts are not executed

By default, PowerShell enforces a restricted execution policy that prevents executing scripts.

To avoid this, use the Puppet-supported PowerShell or specify the appropriate execution policy in the PowerShell command, for example:

```
exec { 'test':
  command => 'powershell.exe -executionpolicy remotesigned -file C:
\test.ps1',
  path      => $::path
}
```

Services are referenced by display names instead of short names

Windows services support a short name and a display name, but Puppet uses only short names.

Verify that your Puppet manifests use the short names, such as `wuauserv` instead of `Automatic Updates`.

Error messages

These are some error messages you might encounter when using Puppet on Windows nodes.

Forge connection or SSL certificate errors

Errors include 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, and The CA bundle included with OpenSSL may not be valid or up to date.

These errors 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, and new 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 then manually install it on the agent node 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.

Make sure to run the `.msi` installation from an elevated `cmd.exe` process. For more information, refer to [Commands with elevated privileges](#) on page 30.

Cannot run on Microsoft Windows without the <GEM_NAME> gem.

This error occurs if you attempt to run Windows without required gems.

Required gems include: `sys-admin`, `win32-process`, `win32-dir`, `win32-service` and `win32-taskscheduler`

Run this command to install the specified gems: `gem install <GEM_NAME>`

/Stage/main]//Scheduled_task[task_system]: Could not evaluate: The operation completed successfully.

This error occurs when the task scheduler gem has a version earlier than 0.2.1.

Run this command to update the task scheduler gem: `gem update win32-taskscheduler`

/Stage/main]//Exec[C:/tmp/<FILE_NAME>.exe]/returns: change from notrun to 0 failed: CreateProcess() failed: Access is denied.

This error occurs when a request for an executable on a remote primary server can't be executed.

Make sure the user and group executable bits are set appropriately on the primary server, for example:

```
file { "C:/tmp/<FILE_NAME>.exe":
  source => "puppet:///modules/<FOLDER_NAME>/<FILE_NAME>.exe",
}

exec { 'C:/tmp/<FILE_NAME>.exe':
  logoutput => true
}
```

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 agent has an incorrect reverse DNS entry.

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 might not be set. For more information, refer to [Microsoft's DNS documentation](#).

Could not request certificate: The certificate retrieved from the primary does not match the agent's private key.

This error can occur if the agent is running in two different security contexts or if the agent's SSL directory is deleted after it retrieves a certificate from the primary server.

Make sure you elevate privileges by selecting **Run as Administrator** when you select **Start Command Prompt with Puppet**.

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 time on the Windows agents isn't synchronized.

Windows agents that are part of an Active Directory (AD) domain automatically have their time synchronized with AD.

For agents that are not part of an AD domain, you must run the following commands to manually enable and add the Windows time service:

```
w32tm /register
net start w32time
w32tm /config /manualpeerlist:<NTP_SERVER> /syncfromflags:manual /update
w32tm /resync
```

Could not parse for environment production: Syntax error at '='; expected '}'

This error occurs if you run `puppet apply -e` from the command line, and the supplied command is surrounded with single quotes (''). The single quotes cause cmd.exe to interpret any rocket hash (=>) in the command as a redirect.

Retry the command with double quotes ("") instead of single quotes.

Logging and debugging

The Windows Event Log can be helpful when troubleshooting issues with Windows nodes.

To enable the Puppet agent to emit `--debug` and `--trace` messages to the Windows Event Log, run this command to stop and restart the Puppet service:

```
c:\>sc stop puppet && sc start puppet --debug --trace
```

Restriction: This setting applies only until the next time the service is restarted or the system is rebooted.

Related information

[Logging for Puppet agent on Windows systems](#)

[Log files installed](#) on page 121

The software distributed with PE generates log files that you can collect for compliance or use for troubleshooting.