Puppet Enterprise 2.0 User's Guide

(Generated on May 29, 2012, from git revision f0db9a496d96dd0afea10fa3ee4dcebb15d810f2)

PE 2.0 » Index

Puppet Enterprise User's Guide

This is the user's guide for Puppet Enterprise 2.0. Use the navigation to the right to move between this guide's sections and chapters, or pick a section below.

- Welcome to Puppet Enterprise

  Introductory info about the components and organization of Puppet Enterprise.

  - Getting Started
  - Components and Roles
  - New Features and Release Notes
  - Known Issues
  - Getting Support

- Installing Puppet Enterprise

  Installation and upgrade instructions, and a short tour of PE's files, users, and services.

  - System Requirements
  - Preparing to Install
  - Basic Installation
  - Upgrading
  - Uninstalling
  - Automated Installation
  - Answer File Reference
  - What gets installed where?

- The Console

  A tour of PE's web console and its new live management features.

  - Accessing the Console
  - Navigating the Console
  - Viewing Reports and Inventory Data
  - Grouping and Classifying Nodes
  - Live Management
  - Live Management: Managing Resources
  - Live Management: Controlling Puppet
  - Live Management: Advanced Tasks

- Puppet For New PE Users

A whirlwind introduction to Puppet; after reading this section, you will be able to create simple Puppet modules and apply them to your nodes. Also contains links to more detailed resources.

  - Overview
  - Your First Module
  - Assigning a Class to a Node
  - Next Steps

- Orchestration For New PE Users

  A whirlwind introduction to orchestration; after reading this section, you will be able to invoke orchestration agents from the command-line client.

  - Overview
  - Usage and Examples

- Cloud Provisioning

  Documentation for PE's cloud provisioner tool, which can create and provision new nodes from the command line.

  - Overview
  - Configuring and Troubleshooting
  - Provisioning with VMware
  - Provisioning with AWS
  - Classifying Nodes and Remotely Installing PE
  - Man Page: puppet node_vmware
  - Man Page: puppet node_aws
  - Man Page: puppet node

- The Compliance Workflow

  A guide to PE's compliance and auditing workflow.

  - Basics and UI
  - Using the Compliance Workflow
  - Tutorial

- The PE Accounts Module

  Documentation for the `pe_accounts` module, which provides a convenient shortcut for managing user accounts.

  - The `pe_accounts::user` Type

- The `pe_accounts` Class

- Maintenance and Troubleshooting

  Help with common tasks and problems.

  - Common Configuration Errors
  - Console Maintenance Tasks
  - Installing Additional Components
  - Reconfiguring Complex Settings

PE 2.0 » Welcome » Getting Started

Index — Welcome: Components and Roles →

Getting Started

Thank you for choosing Puppet Enterprise, the best-of-breed distribution for the Puppet family of systems automation tools.

About This Guide

This guide will help you start using Puppet Enterprise 2.0, and will serve as a reference as you gain more experience with it. It covers PE-specific features, and offers brief introductions to Puppet and MCollective. Use the navigation to the right to move between the guide's sections and chapters.

New Users

If you've never used Puppet before and have just installed Puppet Enterprise, you should read about and experiment with the console's no-code-needed live management features, then follow the tutorial in the "Puppet For New PE Users" section to build your first Puppet module. To get even more out of PE, we recommend the Learning Puppet series and the MCollective documentation.

About Puppet Enterprise

Puppet Enterprise starts with a full-featured, production-scale Puppet stack, then enhances it with the MCollective orchestration framework; a web-based console UI for managing Puppet and editing your systems on the fly; and a cloud provisioning tool for creating and configuring new VM instances.

Licensing

Puppet Enterprise can be evaluated with a complementary ten-node license; beyond that, a commercial per-node license is required for use. A license key file will have been emailed to you after your purchase, and the puppet master will look for this key at `/etc/puppetlabs/license.key`. Puppet will log warnings if the license is expired or exceeded, and you can view the status of your license by running `puppet license` at the command line on the puppet master.

To purchase a license, please see the Puppet Enterprise pricing page, or contact Puppet Labs at sales@puppetlabs.com or (877) 575-9775. For more information on licensing terms, please see the licensing FAQ. If you have misplaced or never received your license key, please contact sales@puppetlabs.com.

About Puppet

Puppet is the leading open source configuration management tool. It allows system configuration "manifests" to be written in a high-level DSL, and can compose modular chunks of configuration to create a machine's unique catalog. Puppet Enterprise implements a client/server Puppet environment, where agent nodes request catalogs from a central puppet master.

About Orchestration

Puppet Enterprise now ships with distributed task orchestration features. Nodes managed by PE will listen for commands over a message bus, and independently take action when they hear an authorized request. This lets you investigate and command your infrastructure in real time without relying on a central inventory.

PE's orchestration features are driven by the MCollective framework and the ActiveMQ message server, and are the backbone of the web console's live management features.

About the Console

Puppet Enterprise's console is the web front-end for managing your systems. The console can:

- Trigger immediate Puppet runs on an arbitrary subset of your nodes
- Browse and edit resources on your nodes in real time
- Analyze reports to help visualize your infrastructure over time
- Browse inventory data and backed-up file contents from your nodes
- Group similar nodes and control the Puppet classes they receive in their catalogs
- Run advanced tasks powered by MCollective plugins

About the Cloud Provisioner

The cloud provisioner is a command line tool for building new nodes. It can create new VMware and Amazon EC2 instances, install Puppet Enterprise on any virtual or physical machine, and classify newly provisioned nodes within your Puppet infrastructure.

Index — Welcome: Components and Roles →

PE 2.0 » Welcome » Components and Roles

← Welcome: Getting Started — Index — Welcome: New Features and Release Notes →

Components and Roles

Puppet Enterprise's features are divided into four main roles, any or all of which can be installed on a single system:

The Puppet Agent Role

This role should be installed on every node you plan to manage with Puppet Enterprise. Nodes with the puppet agent role:

- Run the puppet agent daemon, which pulls configurations from the puppet master and applies them.
- Listen for MCollective messages, and invoke MCollective agent actions when they receive a valid command.
- Report changes to any resources being audited for PE's compliance workflow.

The Puppet Master Role

This role should be installed on one node, which should be a robust, dedicated server; see the system requirements for more detail. The puppet master server:

- Serves configuration catalogs on demand to puppet agent nodes.
- Routes MCollective messages through its ActiveMQ server.
- Can issue valid MCollective commands (from an administrator logged in as the `peadmin` user).

The Console Role

This role should be installed on one node. The console can be installed on the same server as the puppet master, or they can run on separate servers. The console server:

- Serves the console web interface, with which administrators can directly edit resources on nodes, trigger immediate Puppet runs, group and assign classes to nodes, view reports and graphs, view inventory information, approve and reject audited changes, and invoke MCollective agent actions.
- Collects reports from and serves node definitions to the puppet master

The Cloud Provisioner Role

This role should be installed on a secure node to which administrators have shell access. It installs the cloud provisioner tool, which allows a user to:

- Create new VMware and Amazon EC2 virtual machine instances.
- Install Puppet Enterprise on any virtual or physical system.
- Add newly provisioned nodes to a group in the console.

PE 2.0 » Welcome » New Features and Release Notes

New Features and Release Notes

Puppet Enterprise 2.0

2.0 was a major new release of Puppet Enterprise, which introduced the following new features:

Live Management

PE's web console now lets you edit and command your infrastructure in real time. Visit the console's live management tab to:

- Browse resources on your nodes in real-time
- Clone resources to make a group of nodes resemble a known good node
- Trigger puppet runs on an arbitrary set of nodes
- Run advanced MCollective tasks from your browser

Live management works out of the box, without writing any Puppet code.

Cloud Provisioning

PE 2 ships with new command-line tools for building new nodes. From the comfort of your terminal, you can create new machine instances, install PE on any node, and assign new nodes to your existing console groups.

More Secure Console

PE's web console is now served over SSL, and requires a login for access. This lets you control access to the console without having to restrict access by host.

Puppet 2.7

Puppet Enterprise is now built around Puppet 2.7, which made several significant improvements and changes to the Puppet core:

- Puppet can now manage network devices with the vlan, interface, and router resource types.
- There's a new API for creating subcommands called Faces, and Puppet ships with prebuilt subcommands that expose core subsystems at the command line.
- Error messages have been generally improved, including the infamous OpenSSL "Hostname was not match" error.
- Service init scripts are now assumed to have status commands; use `hasstatus => false` to emulate the behavior from 2.6 and earlier.
- Dynamically scoped variable lookup now causes warnings to be logged. If you're getting these warnings, you should begin switching to fully qualified variable names and parameterized classes to eliminate dynamic scoping in your manifests.

See the Puppet release notes for more details.

Other Changes

DASHBOARD IS NOW CONSOLE

What was Puppet Dashboard is now just “the console.”

CHANGES TO ORCHESTRATION FEATURES

- Orchestration is enabled by default for all PE nodes.
- Orchestration tasks can now be invoked directly from the console, with the “advanced tasks” section of the live management page. PE’s orchestration framework also powers the other live management features.
- The `mco` user account on the puppet master is gone, in favor of a new `peadmin` user. This user can still invoke orchestration tasks across your nodes, but it will also gain more general purpose capabilities in future versions.
- PE now includes the `puppetral` plugin, which lets you use Puppet’s Resource Abstraction Layer (RAL) in orchestration tasks.
- For performance reasons, the default message security scheme has changed from AES to PSK.
- The network connection over which messages are sent is now encrypted using SSL.

IMPROVED AND SIMPLIFIED INSTALL EXPERIENCE

The installer asks fewer and smarter questions.

BUILT-IN PUPPET MODULES HAVE BEEN RENAMED

The `mcollectivepe`, `accounts`, and `baselines` modules from PE 1.2 were renamed (to `pe_mcollective, pe_accounts,` and `pe_compliance`, respectively) to avoid namespace conflicts and make their origin more clear. The PE upgrader can install wrapper modules to preserve functionality if you used any of these modules by their previous names.

Puppet Enterprise 2.0.1

This is a maintenance release in the Puppet Enterprise 2.0 series.

Fixed Breakage on Enterprise Linux 6.1 and 6.2

PE 2.0 would often fail during installation on enterprise Linux 6.1 and 6.2 systems (RHEL, CentOS, Oracle Linux, and Scientific Linux). This was due to memory corruption issues with our version of Apache and OpenSSL. The problem has been fixed, and EL > 6.0 systems can now be used as puppet masters and console servers.

Fixed Security Issue: XSS Vulnerability in Console (CVE-2012-0891)

The upstream Puppet Dashboard code used in PE’s web console was found to be vulnerable to cross-site scripting attacks due to insufficient sanitization of user input. See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-0891.

Install and Upgrade Improvements

We've fixed a lot of glitches in PE's installer and upgrader, and done some things to make the experience more user-friendly. Highlights include:

- PE now ships with an uninstaller script.
- The installer will now warn you if you attempt to install a puppet master or console server with less than the required 1GB of memory.
- More secure console and MySQL passwords can be chosen, as the installer will accept a wider array of non-alphanumeric characters. You can also use non-alphanumeric characters in database names.
- We've eliminated a possible source of file permission errors by explicitly setting the umask used by the installer. (This would occasionally cause mysterious installation failures.)
- Upgrading from PE 1.x no longer requires manual edits to `passenger-extra.conf`.
- Answer files created during installation are handled more securely, and are no longer saved as world- or group-readable.
- Handling of remote MySQL databases for the console has been greatly improved. The installer now aborts safely if it isn't able to verify the database credentials, and fixes an issue where the inventory service couldn't use a remote DB.

Support Script

PE now includes an information-gathering script, which can help Puppet Labs support to resolve issues faster. The script is simply called "`support`," and is in the root of the installation tarball, alongside the installer, uninstaller, and upgrader scripts.

When it finishes running, the support script will print the location of its results, which can be sent to Puppet Labs for inspection. This may include sensitive information about your site, so we advise you to examine the collected data before sending it.

Updated Software Versions

We've updated the following software in the PE distribution:

- Puppet has been updated to 2.7.9 (from 2.7.6). See the Puppet release notes for more details. The major improvements include:
  - Faster recursive directory traversal
  - Types and providers can be used during the run in which they are first delivered
- Facter has been updated to 1.6.4, which fixes several Solaris issues.
- PE's web console is now built atop Puppet Dashboard 1.2.4, which fixes an issue with orphaned database records and improves navigation and accessibility.
- Ruby has been updated to 1.8.7 patch level 357. This is due to a security vulnerability found in Ruby. (See here and here.)
- The Rack middleware has been updated to 1.1.3 to respond to the same security vulnerability.
- Apache has been updated to address the following security vulnerabilities:

- CVE-2011-3192
- CVE-2011-3348
- CVE-2011-3368

- Augeas has been updated to the latest upstream version (0.10.0) and modified to handle the correct locations of PE's config files.

All PE Libraries Are Now Namespaced

On RPM-based systems, some of the libraries provided by PE's packages weren't being namespaced, and would cause problems when other software tried to reference system packages that provide those libraries. This has been fixed.

Other Changes

COMPLIANCE FEATURES NOW WORK WITH SOLARIS AGENTS

A quirk in Solaris's cron implementation was preventing the compliance reporting job from running. This has been fixed.

THE ACTIVEMQ HEAP SIZE IS NOW TUNABLE

This is an advanced feature, and should be ignored by most users.

Use the `activemq_heap_mb` parameter to configure this, and see the internal documentation of the `pe_mcollective` module for more details.

Puppet Enterprise 2.0.2

This is a regression-fix release that fixes two issues introduced in PE 2.0.1. Only Debian/Ubuntu and Enterprise Linux 4 systems are affected by this release; in all other respects, it is identical to PE 2.0.1.

Fix Puppet Help Breakage on Debian/Ubuntu Systems

The puppet help subcommand was malfunctioning on Debian and Ubuntu systems. This problem has been fixed.

Fix Upgrade Failures on Enterprise Linux 4 Systems

Agent nodes running RHEL 4 and CentOS 4 were unable to upgrade to PE 2.0.1 due to a packaging error. These nodes can instead upgrade to PE 2.0.2.

Puppet Enterprise 2.0.3

This is a security and bug-fix release in the PE 2.0 series. It patches two security vulnerabilities and fixes a handful of inaccurate hardware facts on Linux.

Security Fix: Group IDs Leak to Forked Processes (CVE-2012-1053)

When executing commands as a different user, Puppet was leaving the forked process with

Puppet's own group permissions. Specifically:

- Puppet's primary group (usually root) was always present in a process's supplementary groups.
- When an `exec` resource had a specified `user` attribute but not a `group` attribute, Puppet would set its effective GID to Puppet's own GID (usually root).
- Permanently changing a process's UID and GID wouldn't clear the supplementary groups, leaving the process with Puppet's own supplementary groups (usually including root).

This caused any untrusted code executed by a Puppet exec resource to be given unexpectedly high permissions. See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-1053.

Security Fix: k5login Type Writes Through Symlinks (CVE-2012-1054)

If a user's `.k5login` file was a symlink, Puppet would overwrite the link's target when managing that user's login file with the k5login resource type. This allowed local privilege escalation by linking a user's `.k5login` file to root's `.k5login` file.

See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-1054.

Bug Fix: Inaccurate Hardware Facts

In previous versions of PE, the following facts were often inaccurate on Linux systems that lacked the pciutils/pmtools and dmidecode packages:

- `virtual`
- `is_virtual`
- `manufacturer`
- `productname`
- `serialnumber`

PE now lists the necessary packages as prerequisites for Facter, which makes these facts more reliable.

Upgrader Improvements

The upgrader now skips unnecessary steps for upgrades from 2.0.x versions.

Puppet.conf is No Longer World-Readable By Default

The `/etc/puppetlabs/puppet/puppet.conf` file on the puppet master is now created with default permissions of `0600`, to improve security in cases where the puppet master is a multi-purpose server with untrusted user accounts.

Apache Now Discards X-Forwarded-For Headers

Puppet Enterprise's Apache configuration now discards any X-Forwarded-For header from requests before passing them to Puppet. In rare configurations, Ruby and Rack's handling of this header could allow agents to impersonate each other when making unauthenticated requests.

← Welcome: Components and Roles — Index — Welcome: Known Issues →

PE 2.0 » Welcome » Known Issues

← Welcome: New Features and Release Notes — Index — Welcome: Getting Support →

Known Issues in Puppet Enterprise 2.0

As we discover them, this page will be updated with known issues in each maintenance release of Puppet Enterprise 2.0. If you find new problems yourself, please file bugs in Puppet here and bugs specific to Puppet Enterprise here.

To find out which of these issues you are affected by, run `/opt/puppet/bin/puppet --version`, the output of which will look something like `2.7.9 (Puppet Enterprise 2.0.1)`. To upgrade to a newer version of Puppet Enterprise, see the chapter on upgrading.

Issues Still Outstanding

The following issues affect the currently shipped version of PE and all prior releases in the 2.0.x series, unless otherwise stated.

Upgrades May Fail With MySQL Errors

Several users have encountered failures when upgrading to PE 2.0.3, and there have been reports of similar failures when running previous upgrades. These failures:

- Are limited to the console server
- Usually affect sites with a very large console database
- Are triggered by a MySQL error when running database migrations

The upgrader's output in these cases resembles the following:

```
(in /opt/puppet/share/puppet-dashboard)
== AddReportForeignKeyConstraints: migrating ================================
Going to delete orphaned records from metrics, report_logs, resource_statuses,
resource_events
Preparing to delete from metrics
2012-01-27 17:51:31: Deleting 0 orphaned records from metrics
Deleting 100%
|############################################################################| Time:
00:00:00
Preparing to delete from report_logs
2012-01-27 17:51:31: Deleting 0 orphaned records from report_logs
Deleting 100%
|############################################################################| Time:
00:00:00
Preparing to delete from resource_statuses
2012-01-27 17:51:31: Deleting 0 orphaned records from resource_statuses
Deleting 100%
|############################################################################| Time:
00:00:00
Preparing to delete from resource_events
2012-01-27 17:51:31: Deleting 0 orphaned records from resource_events
Deleting 100%
|############################################################################| Time:
```

```
00:00:00
-- execute("ALTER TABLE reports ADD CONSTRAINT fk_reports_node_id FOREIGN KEY
(node_id) REFERENCES nodes(id) ON DELETE CASCADE;")
rake aborted!
An error has occurred, all later migrations canceled:
Mysql::Error: Can't create table 'console.#sql-328_ff6' (errno: 121): ALTER
TABLE reports ADD CONSTRAINT fk_reports_node_id FOREIGN KEY (node_id)
REFERENCES nodes(id) ON DELETE CASCADE;
(See full trace by running task with --trace)
================================================================================

!! ERROR: Cancelling installation
================================================================================
```

The cause of these failures is still under investigation, but it appears to involve a too-small lock table, which is governed by the `innodb_buffer_pool_size` setting in the MySQL server configuration.

WORKAROUND

Until the upgrader can handle these cases by itself, we recommend the following:

If you haven't yet upgraded PE on your console server:

- Edit the MySQL config file on your database server (usually located at `/etc/my.cnf`, but your system may differ) and set the value of `innodb_buffer_pool_size` to at least `80M`. (Its default value is 8 MB, or 8388608 bytes.)

  Example diff:

```
 [mysqld]
 datadir=/var/lib/mysql
 socket=/var/lib/mysql/mysql.sock
 user=mysql
 # Default to using old password format for compatibility with mysql 3.x
 # clients (those using the mysqlclient10 compatibility package).
 old_passwords=1
+innodb_buffer_pool_size = 80M

 # Disabling symbolic-links is recommended to prevent assorted security risks;
 # to do so, uncomment this line:
 # symbolic-links=0

 [mysqld_safe]
 log-error=/var/log/mysqld.log
 pid-file=/var/run/mysqld/mysqld.pid
```

- Restart the MySQL server:

```
# sudo /etc/init.d/mysqld restart
```

- Perform a normal upgrade of Puppet Enterprise on the console server.

If you have already suffered a failed upgrade:

- On your database server, log into the MySQL client as either the root user or the console user:

```
# mysql -u console -p
Enter password: <password>
```

- Execute the following SQL statements:

```
USE console
ALTER TABLE reports DROP FOREIGN KEY fk_reports_node_id;
ALTER TABLE resource_events DROP FOREIGN KEY
fk_resource_events_resource_status_id;
ALTER TABLE resource_statuses DROP FOREIGN KEY
fk_resource_statuses_report_id;
ALTER TABLE report_logs DROP FOREIGN KEY fk_report_logs_report_id;
ALTER TABLE metrics DROP FOREIGN KEY fk_metrics_report_id;
```

- Edit the MySQL config file and restart the MySQL server, as described above.
- Re-run the upgrader, which should now finish successfully.

For more information about the lock table size, see this MySQL bug report.

On Linode/Xen Instances, Facter Always Prints Error Messages

(Issue #12813)

This issue was introduced in PE 2.0.3.

On Linode instances, and possibly other Xen platforms, Facter prints the following harmless but annoying messages to STDERR every time Puppet runs:

```
pcilib: Cannot open /proc/bus/pci
lspci: Cannot find any working access method.
```

This will be fixed in the next patch release of PE 2.0. For now, the noise can be suppressed by directly patching the `/opt/puppet/lib/ruby/site_ruby/1.8/facter/virtual.rb` file, as shown below:

```
diff --git lib/facter/virtual.rb lib/facter/virtual.rb
index e617359..94b10c5 100644
--- /opt/puppet/lib/ruby/site_ruby/1.8/facter/virtual.rb
```

```
+++ /opt/puppet/lib/ruby/site_ruby/1.8/facter/virtual.rb
@@ -89,7 +89,7 @@ Facter.add("virtual") do
     end

     if result == "physical"
-      output = Facter::Util::Resolution.exec('lspci')
+      output = Facter::Util::Resolution.exec('lspci 2>/dev/null')
       if not output.nil?
         output.each_line do |p|
           # --- look for the vmware video card to determine if it is virtual
=> vmware.
```

`pe-httpd` Must Be Restarted After Revoking Certificates

(Issue #8421)

Due to an upstream bug in Apache, the `pe-httpd` service on the puppet master must be restarted after revoking any node's certificate.

After using `puppet cert revoke` or `puppet cert clean` to revoke a certificate, restart the service by running:

```
# sudo /etc/init.d/pe-httpd restart
```

Installer Cannot Prevent or Recover From DNS Errors

The installer in PE 2.0 does not currently check for DNS misconfiguration. Such problems can cause the installer to fail, leaving the PE software in an undefined state. To work around this, you should:

- Be sure to read this guide's Preparing to Install chapter before installing, and make sure DNS at your site is configured appropriately.
- If the installer has failed, follow the instructions in the troubleshooting section of this guide.

Internet Explorer 8 Can't Access Live Management Features

The console's live management page doesn't load in Internet Explorer 8. Although we are working on supporting IE8, you should currently use another browser (such as Internet Explorer 9 or Google Chrome) to access PE's live management features.

Dynamic Man Pages are Incorrectly Formatted

Man pages generated with the `puppet man` subcommand are not formatted as proper man pages, and are instead displayed as Markdown source text. This is a purely cosmetic issue, and the pages are still fully readable.

Issues Affecting PE 2.0.2

Incorrect Hardware Facts

This issue was fixed in PE 2.0.3.

On Linux systems where the pciutils, pmtools, or dmidecode packages are not installed, the `virtual`, `is_virtual`, `manufacturer`, `productname`, and `serialnumber` facts are frequently inaccurate. In PE 2.0.3 and later, these facts are kept accurate by requiring the necessary packages from the OS's repository.

Security Issue: Group IDs Leak to Forked Processes (CVE-2012-1053)

This issue was fixed in PE 2.0.3. It affected PE versions between 1.0 and 2.0.2.

When executing commands as a different user, Puppet leaves the forked process with Puppet's own group permissions. Specifically:

- Puppet's primary group (usually root) is always present in a process's supplementary groups.
- When an `exec` resource has a specified `user` attribute but not a `group` attribute, Puppet will set its effective GID to Puppet's own GID (usually root).
- Permanently changing a process's UID and GID won't clear the supplementary groups, leaving the process with Puppet's own supplementary groups (usually including root).

This causes any untrusted code executed by a Puppet exec resource to be given unexpectedly high permissions. See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-1053.

Security Issue: k5login Type Writes Through Symlinks (CVE-2012-1054)

This issue was fixed in PE 2.0.3. It affected PE versions between 1.0 and 2.0.2.

If a user's `.k5login` file is a symlink, Puppet will overwrite the link's target when managing that user's login file with the k5login resource type. This allows local privilege escalation by linking a user's `.k5login` file to root's `.k5login` file.

See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-1054.

Security Issue: Puppet.conf on Puppet Master is World-Readable

This issue was fixed in PE 2.0.3.

The `/etc/puppetlabs/puppet/puppet.conf` file on the puppet master is world-readable by default (permissions mode `0644`), which can expose the console's database password to unauthorized users.

Security Issue: X-Forwarded-For Headers Are Respected for Unauthenticated Requests

This issue was fixed in PE 2.0.3. It affected PE versions between 1.0 and 2.0.2.

By default, Puppet's unauthenticated services (certificate requests, fetching the CA certificate…) are not restricted by hostname, but `auth.conf` makes it possible to open extra services to unauthenticated connections and restrict those connections by hostname. If the puppet master were configured like this, it would be possible for nodes to impersonate other nodes simply by modifying the X-Forwarded-For header in their request.

This rare possibility was removed in PE 2.0.3 by changing the Apache configuration to discard any X-Forwarded-For headers from requests before passing them to Puppet.

Uninstaller Cannot Remove Databases if MySQL's Root Password Has Special Characters

This issue was fixed in PE 2.0.3.

If your database server's root password contains certain non-alphanumeric characters, the uninstaller may not be able to log in and delete PE's databases, and you will have to remove them manually. This issue was present when the uninstaller was introduced in PE 2.0.1.

Issues Affecting PE 2.0.1

EL 4 Agent Nodes Cannot Upgrade from 2.0.0 to 2.0.1

This issue was fixed in PE 2.0.2.

Due to a packaging error, Enterprise Linux 4 agent nodes running PE 2.0.0 cannot be upgraded to PE 2.0.1.

Puppet Help Broken on Debian/Ubuntu Systems

This issue was fixed in PE 2.0.2.

Due to a packaging error, the puppet help subcommand malfunctions on Debian and Ubuntu systems, resulting in errors like the following:

```
puppet help cert list
err: RubyGem version error: excon(0.6.5 not ~> 0.7.3)

err: Try 'puppet help help help' for usage
```

Issues Affecting PE 2.0.0

Puppet Master and Console Roles Break Under EL 6.1 and 6.2

This issue was fixed in PE 2.0.1.

The versions of Apache and OpenSSL used in PE 2.0.0 suffer from memory corruption under any enterprise Linux 6 system (RHEL, CentOS, Oracle Linux, and Scientific Linux) later than version 6.0. This would cause dramatic failures during installation on these OS versions.

Ruby 1.8.7 Patchlevel 302 is Vulnerable to a Denial of Service Attack

This issue was fixed in PE 2.0.1 by upgrading Ruby and Rack.

The version of Ruby used by PE 2.0.0 is vulnerable to a denial of service attack via a predictable hashbucket algorithm. See here and here for more details.

Apache Contains Several Security Vulnerabilities

Apache was upgraded to address these issues in PE 2.0.1.

The version of Apache used by PE 2.0.0 is vulnerable to the following CVE issues:

- CVE-2011-3192
- CVE-2011-3348
- CVE-2011-3368

Security Issue: XSS Vulnerability in Puppet Dashboard (CVE-2012-0891)

This issue was fixed in PE 2.0.1.

The upstream Puppet Dashboard code used in PE's web console was found to be vulnerable to cross-site scripting attacks due to insufficient sanitization of user input. See here for more details, including hotfixes for previous versions of PE.

This vulnerability has a CVE identifier of CVE-2012-0891.

Installer Cannot Detect or Recover From a Misconfigured Firewall

This issue was fixed in PE 2.0.1, which will detect and warn of firewall misconfigurations.

The installer in PE 2.0.0 may fail and leave the PE software in an undefined state if your firewall doesn't allow access to the required ports. If your installation fails due to firewall issues, you should follow the instructions in the troubleshooting section of this guide.

Console's `reports:prune` Task Leaves Orphaned Data

This issue was fixed in PE 2.0.1, and the upgrade process will delete any orphaned data that remains in your console database without requiring any additional user action.

The console's historical node reports can be deleted periodically to control disk usage; this is usually done by creating a cron job for the `reports:prune` rake task. However, in PE 2.0.0, this task leaves behind orphaned records in the database.

If you aren't at risk of running out of disk space, simply upgrade to Puppet Enterprise 2.0.1 or later at your leisure.

If you are about to run out of disk, and cannot immediately upgrade PE on your console server, you can:

- Download this updated rake fragment.

- Put it in `/opt/puppet/share/puppet-dashboard/lib/tasks/` on your console server, replacing the existing `prune_reports.rake` file.
- Run the following command on your console server:

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
reports:prune:orphaned
```

  This task will have to be run after every time you run the `reports:prune` task, until you are able to upgrade to an unaffected version of Puppet Enterprise.

Answer Files Created During Installation are Saved as World-Readable

This issue was fixed in PE 2.0.1.

Answer files created when installing PE 2.0.0 are saved as world- and group-readable. These files may contain sensitive information, and should be either deleted or given more restrictive permissions.

The Uninstaller Script is Not Shipped With PE

This issue was fixed in PE 2.0.1, which includes the uninstaller.

The Puppet Enterprise uninstaller script was not included with PE 2.0. Although it is included in subsequent PE releases, you can download it here.

Before you can use it, you must move the uninstaller script into the directory which contains the installer script. The uninstaller and the installer must be in the same directory. Once it is in place, you can make the uninstaller executable and run it:

```
# sudo chmod +x puppet-enterprise-uninstaller
# sudo ./puppet-enterprise-uninstaller
```

When Upgrading, `passenger-extra.conf` Requires Manual Edits

This issue was fixed in PE 2.0.1, and the upgrader script now automatically tunes the `passenger-extra.conf` file.

Upgrading to PE 2.0.0 requires you to edit the `/etc/puppetlabs/httpd/conf.d/passenger-extra.conf` file, on both the puppet master and the console server, so that it looks like this:

```
# /etc/puppetlabs/httpd/conf.d/passenger-extra.conf
PassengerHighPerformance on
PassengerUseGlobalQueue on
PassengerMaxRequests 40
```

```
PassengerPoolIdleTime 15
PassengerMaxPoolSize 8
PassengerMaxInstancesPerApp 4
```

Some of these settings can be tuned:

- `PassengerMaxPoolSize` should be four times the number of CPU cores in the server.
- `PassengerMaxInstancesPerApp` should be one half the `PassengerMaxPoolSize`.

Compliance Features Don't Work with Solaris Agents

This issue was fixed in PE 2.0.1.□

A quirk in Solaris's cron implementation prevents the compliance reporting job from running on PE 2.0.0. This means resources cannot be audited on Solaris agents without manually repairing the cron job or upgrading to PE 2.0.1.

Libraries Provided by PE Aren't Namespaced on RPM-based Systems

This issue was fixed in PE 2.0.1.□

On RPM-based systems, some of the libraries provided by PE 2.0.0's packages are not namespaced. This can block proper dependency resolution when installing software that expects the system-provided versions of those libraries.

← Welcome: New Features and Release Notes — Index — Welcome: Getting Support →

PE 2.0 » Welcome » Getting Support

← Welcome: Known Issues — Index — Installing: System Requirements →

Getting Support for Puppet Enterprise

Getting support for Puppet Enterprise is easy, both from Puppet Labs and the community of Puppet Enterprise users. We provide responsive, dependable, quality support to resolve any issues regarding the installation, operation and use of Puppet.

There are three primary ways to get support for Puppet Enterprise:

- Reporting issues to the Puppet Labs customer support portal
- Joining the Puppet Enterprise user group.
- Seeking help from the Puppet open source community.

Reporting Issues to the customer support portal

Paid Support

Puppet Labs provides two levels of commercial support offerings for Puppet Enterprise: Standard and Premium. Both offerings allow you to report your support issues to our confidential customer support portal. You will receive an account and log-on for this portal when you purchase Puppet Enterprise.

Customer support portal: https://support.puppetlabs.com

THE PE SUPPORT SCRIPT

When seeking support, you may be asked to run the information-gathering support script included with in the Puppet Enterprise installer tarball. This script is located in the root of the unzipped tarball and is named simply "`support`."

This script will collect a large amount of system information, compress it, and print the location of the zipped tarball when it finishes running; an uncompressed directory (named `support`) containing the same data will be left in the same directory the compressed copy. We recommend that you examine the collected data before forwarding it to Puppet Labs, as it may contain sensitive information that you will wish to redact.

The information collected by the support script includes:

- iptables info (is it loaded? what are the inbound and outbound rules?) (both ipv4 and ipv6)
- a full run of facter (if installed)
- selinux status
- the amount of free disk and memory on the system
- hostname info (`/etc/hosts` and the output of `hostname --fqdn`)
- the umask of the system
- ntp configuration (what servers are available, the offset from them)

- a listing (no content) of the files in `/opt/puppet`, `/var/opt/lib/pe-puppet` and `/var/opt/lib/pe-puppetmaster`
- the os and kernel
- a list of installed packages
- the current process list
- a listing of puppet certs
- a listing of all services (except on Debian, which lacks the equivalent command)
- current environment variables
- whether the puppet master is reachable
- the output of `mco ping` and `mco inventory`

It also copies the following files:

- system logs
- the contents of `/etc/puppetlabs`
- the contents of `/var/log/pe-*`

Free Support

If you are evaluating Puppet Enterprise, we also offer support during your evaluation period. During this period you can report issues with Puppet Enterprise to our public support portal. Please be aware that all issues filed here are viewable by all other users.

Public support portal: http://projects.puppetlabs.com/projects/puppet-enterprise

Join the Puppet Enterprise user group

http://groups.google.com/a/puppetlabs.com/group/pe-users

- Click on “Sign in and apply for membership”
- Click on “Enter your email address to access the document”
- Enter your email address.

Your request to join will be sent to Puppet Labs for authorization and you will receive an email when you’ve been added to the user group.

Getting support from the existing Puppet Community

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

- Puppet open source user group:

  http://groups.google.com/group/puppet-users

- Puppet Developers group:

  http://groups.google.com/group/puppet-dev
- Report issues with the open source Puppet project:

  http://projects.puppetlabs.com/projects/puppet

← Welcome: Known Issues — Index — Installing: System Requirements →

PE 2.0 » Installing » System Requirements

← Welcome: Getting Support — Index — Installing: Preparing to Install →

System Requirements

Operating System

Puppet Enterprise 2.0 supports the following systems:

Operating system	Version	Arch	Roles
Red Hat Enterprise Linux	5 and 6	x86 and x86_64	all roles
Red Hat Enterprise Linux	4	x86 and x86_64	agent
CentOS	5 and 6	x86 and x86_64	all roles
CentOS	4	x86 and x86_64	agent
Ubuntu LTS	10.04	32- and 64-bit	all roles
Debian	Lenny (5) and Squeeze (6)	i386 and amd64	all roles
Oracle Linux	5 and 6	x86 and x86_64	all roles
Scientific Linux	5 and 6	x86 and x86_864	all roles
SUSE Linux Enterprise Server	11	x86 and x86_864	all roles
Solaris	10	SPARC and x86_64	agent

Hardware

Puppet Enterprise's hardware requirements depend on the roles a machine performs.

- The puppet master role should be installed on a robust, dedicated server.
  - Minimum requirements: 2 processor cores, 1 GB RAM, and very accurate timekeeping.
  - Recommended requirements: Physical hardware or Xen or KVM virtual server, with 2–4 processor cores and 4 GB RAM. Performance will vary, but this configuration can generally manage approximately 1,000 agent nodes.

- The console role should usually be installed on the same server as the puppet master, but can optionally be separated.
  - Minimum requirements: A machine able to handle moderate web traffic and perform processor-intensive background tasks.

- The cloud provisioner role has very modest requirements.
  - Minimum requirements: A system which provides interactive shell access for trusted users.

- The puppet agent role has very modest requirements.
  - Minimum requirements: Any hardware able to comfortably run a supported operating system.

PE 2.0 » Installing » Preparing to Install

Preparing to Install

Planning

Before anything else, decide in advance which server will fill the role of puppet master. This should be a robust dedicated server with accurate timekeeping and at least 1 GB of RAM.

You must also decide whether the puppet master server will also be filling the console role. In general, we recommend installing the master and the console together.

Plan to install PE on the puppet master before installing on any agent nodes. If you will be splitting the master and console roles, you should install the console role immediately after the master role.

Configuring Your Site

Before installing Puppet Enterprise at your site, you should make sure that your nodes and network are properly configured.

Name Resolution

- Decide on a preferred name or set of names at which agent nodes should contact the puppet master server.
- Ensure that the puppet master server can be reached via domain name lookup by all of the future puppet agent nodes at the site.

You can also simplify configuration of agent nodes by using a CNAME record to make the puppet master reachable at the hostname `puppet`. (This is the default puppet master hostname that is automatically suggested when installing an agent node.)

Firewall Configuration

Configure your firewalls to accomodate Puppet Enterprise's network traffic. The short version is that you should open up ports 8140, 61613, and 443. The more detailed version is:

- All agent nodes must be able to send requests to the puppet master on ports 8140 (for Puppet) and 61613 (for MCollective).
- The puppet master must be able to accept inbound traffic from agents on ports 8140 (for Puppet) and 61613 (for MCollective).
- Any hosts you will use to access the console must be able to reach the console server on port 443, or whichever port you specify during installation. (Users who cannot run the console on port 443 will often run it on port 3000.)
- If you will be invoking MCollective client commands from machines other than the puppet master, they will need to be able to reach the master on port 61613.
- If you will be running the console and puppet master on separate servers, the console server must be able to accept traffic from the puppet master (and the master must be able to send

requests) on ports 443 and 8140. The Dashboard server must also be able to send requests to the puppet master on port 8140, both for retrieving its own catalog and for viewing archived file contents.

Downloading PE

Before installing Puppet Enterprise, you must download it from the Puppet Labs website.

Choosing Your Installer Tarball

Puppet Enterprise can be downloaded in tarballs specific to your OS version and architecture, or as a universal tarball. Although the universal tarball is simpler to use, it is roughly ten times the size of a version-specific tarball, and may prove unwieldy depending on your installation plan.

AVAILABLE TARBALLS

Filename ends with...	Will install...
`-all.tar`	anywhere
`-debian-<version and arch>.tar`	on Debian
`-el-<version and arch>.tar`	on RHEL, CentOS, Scientific Linux, or Oracle Linux
`-sles-<version and arch>.tar`	on SUSE Linux Enterprise Server
`-solaris-<version and arch>.tar`	on Solaris
`-ubuntu-<version and arch>.tar`	on Ubuntu LTS

← Installing: System Requirements — Index — Installing: Basic Installation →

PE 2.0 » Installing » Basic Installation

← Installing: Preparing to Install — Index — Installing: Upgrading →

Basic Installation

Starting the Installer

To install PE:

- Unarchive the installer tarball, usually with `tar -xzf <INSTALLER TARBALL>`.
- Navigate to the resulting directory in your shell.
- Run the `puppet-enterprise-installer` script with root privileges:

```
# sudo ./puppet-enterprise-installer
```

- Answer the interview questions to customize your installation.
- Log into the puppet master server and sign the new node's certificate.
- If you have purchased PE and are installing the puppet master, copy your license key into place.

The installer can also be run non-interactively; see the chapter on automated installation for details.

Note that after the installer has finished installing and configuring PE, it will save your interview answers to a file called `answers.lastrun`.

Installer Options

The installer will accept the following command-line flags:

`-h`
Display a brief help message.
`-s <ANSWER FILE>`
Save answers to file and quit without installing.
`-a <ANSWER FILE>`
Read answers from file and fail if an answer is missing.
`-A <ANSWER FILE>`
Read answers from file and prompt for input if an answer is missing.
`-D`
Display debugging information.
`-l <LOG FILE>`
Log commands and results to file.
`-n`
Run in 'noop' mode; show commands that would have been run during installation without running them.

Customizing Your Installation

The PE installer configures Puppet by asking a series of questions. Most questions have a default answer (displayed in brackets), which you can accept by pressing enter without typing a replacement. For questions with a yes or no answer, the default answer is capitalized (e.g. "`[y/N]`").

Roles

First, the installer will ask which of PE's roles (puppet master, console, cloud provisioner, and puppet agent) to install. The roles you apply will determine which other questions the installer will ask.

If you choose the puppet master or console roles, the puppet agent role will be installed automatically.

The puppet master and console roles should each be installed on only one system.

Puppet Master Options

CERTNAME

The certname is the puppet master's unique identifier. It should be a DNS name at which the master server can be reliably reached, and defaults to its fully-qualified domain name.

(If the master's certname is not one of its DNS names, you may need to edit puppet.conf after installation.)

VALID DNS NAMES

The master's certificate contains a static list of valid DNS names, and agents won't trust the master if they contact it at an unlisted address. You should make sure that this list contains the DNS name or alias you'll be configuring your agents to contact.

The valid DNS names setting should be a comma-separated list of hostnames. The default set of DNS names will be derived from the certname you chose, and will include the default puppet master name of "puppet."

LOCATION OF THE CONSOLE SERVER

If you are splitting the puppet master and console roles across different machines, the installer will ask you for the hostname and port of the console server.

Console Options

The console is usually run on the same server as the puppet master, but can also be installed on a separate machine. If you are splitting the console and puppet master roles, install the console after the puppet master.

PORT

You must choose a port on which to serve the console's web interface. If you aren't already serving web content from this machine, it will default to port 443, so you can reach it at `https://yourconsoleserver` without specifying a port.

If the installer detects another web server on the node, it will suggest the first open port at or above 3000.

USER NAME AND PASSWORD

As the console's web interface is a major point of control for your infrastructure, access is restricted

with a user name and password. Additional users and passwords can be added later with Apache's standard authentication tools.

The only forbidden characters for a console password are `\` (backslash), `'` (single quote), and `$$` (two dollar signs in a row).

INVENTORY CERTNAME AND DNS NAMES (OPTIONAL)

If you are splitting the master and the console roles, the console will maintain an inventory service to collect facts from the puppet master. Like the master, the inventory service needs a unique certname and a list of valid DNS names.

DATABASE

The console needs a pair of MySQL databases and a MySQL user in order to operate. If a MySQL server isn't already present on this system, the installer can automatically configure everything the console needs; just confirm that you want to install a new database server, and configure the following settings:

- A password for MySQL's root user
- A name for the console's primary database
- A MySQL user name for the console
- A password for the console's user

The only forbidden characters for a database password are `\` (backslash), `'` (single quote), and `$$` (two dollar signs in a row).

If you don't install a new database server, you can either manually create a database and MySQL user for the console and configure the settings above with the correct information, or allow the installer to log into the MySQL server as root and automatically configure the databases.

Note that if you want to automatically configure databases on a remote database server, you must make sure the root MySQL user is allowed to log in remotely.

If you are not automatically configuring the databases, you can create the necessary MySQL resources in a secondary shell session while the installer is waiting for input. The SQL commands you need will resemble the following:

```
CREATE DATABASE console CHARACTER SET utf8;
CREATE DATABASE console_inventory_service CHARACTER SET utf8;
CREATE USER 'console'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON console.* TO 'console'@'localhost';
GRANT ALL PRIVILEGES ON console_inventory_service.* TO 'console'@'localhost';
FLUSH PRIVILEGES;
```

Note that the names of the two databases are related: the name of the inventory service database must start with the name of the primary console database, followed by `_inventory_service`.

Note also that the hostname for the console user will differ if you are using a remote database server.

Consult the MySQL documentation for more info.

A local database is best, but you can also create a database on a remote server. If you do, you'll need to answer `Y` when asked if your MySQL server is remote, and provide:

- The database server's hostname
- The database server's port (default: 3306)

Puppet Agent Options

CERTNAME

The certname is the agent node's unique identifier.

This defaults to the node's fully-qualified domain name, but any arbitrary string can be used. If hostnames change frequently at your site or are otherwise unreliable, you may wish to use UUIDs or hashed firmware attributes for your agent certnames.

PUPPET MASTER HOSTNAME

Agent nodes need the hostname of a puppet master server. This must be one of the valid DNS names you chose when installing the puppet master.

This setting defaults to `puppet`.

Vendor Packages

Puppet Enterprise may need some extra system software from your OS vendor's package repositories. If any of this software isn't yet installed, the installer will list the packages it needs and offer to automatically install them. If you decline, it will exit so you can install the necessary packages manually before installing.

A note about Java and MySQL under Enterprise Linux variants: Java and MySQL packages provided by Oracle can satisfy the puppet master and console roles' Java and MySQL dependencies, but the installer can't make that decision automatically and will default to using the OS's packages. If you wish to use Oracle's packages instead of the OS's, you must first use RPM to manually install the `pe-virtual-java` and/or `pe-virtual-mysql` packages included with Puppet Enterprise:

```
# sudo rpm -ivh packages/pe-virtual-java-1.0-1.pe.el5.noarch.rpm
```

Find these in the installer's `packages/` directory. Note that these packages may have additional ramifications if you later install other software that depends on OS MySQL or Java packages.

Convenience Links

PE installs its binaries in `/opt/puppet/bin` and `/opt/puppet/sbin`, which aren't included in your

default `$PATH`. If you want to make the Puppet tools more visible to all users, the installer can make symlinks in `/usr/local/bin` for the `facter, puppet, puppet-module, pe-man`, and `mco` binaries.

Finishing Up

Signing Agent Certificates

Before nodes with the puppet agent role can fetch configurations or appear in the console, an administrator has to sign their certificate requests. This helps prevent unauthorized nodes from intercepting sensitive configuration data.

During installation, PE will automatically submit a certificate request to the puppet master. Before the agent can retrieve any configurations, a user will have to sign a certificate for it.

Certificate signing is done on the puppet master node. To view the list of pending certificate requests, run:

```
$ sudo puppet cert list
```

To sign one of the pending requests, run:

```
$ sudo puppet cert sign <name>
```

After signing a new node's certificate, it may take up to 30 minutes before that node appears in the console and begins retrieving configurations.

Verifying Your License

When you purchased Puppet Enterprise, you should have been sent a `license.key` file that lists how many nodes you can deploy. For PE to run without logging license warnings, you should copy this file to the puppet master node as `/etc/puppetlabs/license.key`. If you don't have your license key file, please email sales@puppetlabs.com and we'll re-send it.

← Installing: Preparing to Install — Index — Installing: Upgrading →

PE 2.0 » Installing » Upgrading

← Installing: Basic Installation — Index — Installing: Uninstalling →

Upgrading Puppet Enterprise

To upgrade from a previous version of Puppet Enterprise, use the same installer tarball as in a basic installation, but don't run the `puppet-enterprise-installer` script. Instead, run `puppet-enterprise-upgrader`.

Depending on the version you upgrade from, you may need to take extra steps after running the upgrader. See below for your specific version.

Note that if your console database is very large, the upgrader may take a long time on the console node, possibly thirty minutes or more. This is due to a resource-intensive database migration that must be run. Make sure that you schedule your upgrade appropriately, and avoid interrupting the upgrade process.

> **IMPORTANT NOTE**
>
> Due to a known issue that can potentially cause upgrade failures, all users should increase the size of their MySQL server's `innodb_buffer_pool_size` before upgrading the console server node. See here for details and instructions, including how to recover from a failed upgrade.

Checking For Updates

Check here to find out what the latest maintenance release of Puppet Enterprise is. You can run `puppet --version` at the command line to see the version of PE you are currently running.

Downloading PE

See the Preparing to Install chapter of this guide for information on downloading PE.

Starting the Upgrader

The upgrader must be run with root privileges:

```
# ./puppet-enterprise-upgrader
```

This will start the upgrader in interactive mode. If the puppet master role and the console (previously dashboard) role are installed on different servers, you must upgrade the puppet master first.

Note that if your console database is very large, the upgrader may take a long time on the console node, possibly thirty minutes or more. This is due to a resource-intensive database migration that

must be run. Make sure that you schedule your upgrade appropriately, and avoid interrupting the upgrade process.

> **IMPORTANT NOTE**
>
> Due to a known issue that can potentially cause upgrade failures, all users should increase the size of their MySQL server's `innodb_buffer_pool_size` before upgrading the console server node. See here for details and instructions, including how to recover from a failed upgrade.

Upgrader Options

Like the installer, the upgrade will accept some command-line options:

`-h`
Display a brief help message.
`-s <ANSWER FILE>`
Save answers to file and quit without installing.
`-a <ANSWER FILE>`
Read answers from file and fail if an answer is missing. See the upgrader answers section of the answer file reference for a list of available answers.
`-A <ANSWER FILE>`
Read answers from file and prompt for input if an answer is missing. See the upgrader answers section of the answer file reference for a list of available answers.
`-D`
Display debugging information.
`-l <LOG FILE>`
Log commands and results to file.
`-n`
Run in 'noop' mode; show commands that would have been run during installation without running them.

Non-interactive upgrades work identically to non-interactive installs, albeit with different answers available.

Configuring the Upgrade

The upgrader will ask you the following questions:

Cloud Provisioner

PE 2 includes a cloud provisioner tool that can be installed on trusted nodes where administrators have shell access. On every node you upgrade, you'll be asked whether to install the cloud provisioner role.

Vendor Packages

If PE 2.0 needs any packages from your OS's repositories, it will ask permission to install them.

Puppet Master Options

REMOVING `mco`'S HOME DIRECTORY

The `mco` user from PE 1.2 gets deleted during the upgrade, and is replaced with the `peadmin` user.

If the `mco` user had any preference files or documents you need, you should tell the upgrader to preserve the `mco` user's home directory; otherwise, it will be deleted.

INSTALLING WRAPPER MODULES

In PE 2.0, the `mcollectivepe`, `accounts`, and `baselines` modules were renamed to `pe_mcollective, pe_accounts,` and `pe_compliance`, respectively. If you have used any of these modules by their previous names, you should install the wrapper modules so your site will continue to work while you switch over.

Console Options

USER NAME AND PASSWORD

The console, which replaces Puppet Dashboard, now requires a user name and a password for web access. The upgrader will ask you to choose this name and password.

Final Steps: From an Earlier PE 2.0 Release

No extra steps are needed when upgrading between maintenance releases of PE 2.0.

Final Steps: From PE 1.2

No extra steps are needed when upgrading from PE 1.2.x to PE 2.0.1 or later.

Note that some features may not be available until puppet agent has run once on every node. In normal installations, this means all features will be available within 30 minutes after upgrading all nodes.

Final Steps: From PE 1.1 or 1.0

Important note: Upgrades from some configurations of PE 1.1 and 1.0 aren't fully supported. To upgrade from PE 1.1 or 1.0, you must have originally installed the puppet master and Puppet Dashboard roles on the same node. Contact Puppet Labs support for help with other configurations on a case-by-case basis, and see issue #10872 for more information.

After running the upgrader on the puppet master/Dashboard (now console) node, you must:

- Stop the `pe-httpd` service
- Create a new database for the inventory service and grant all permissions on it to the console's MySQL user.
- Manually edit the puppet master's `puppet.conf`, `auth.conf`, `site.pp`, and `settings.yml` files
- Generate and sign certificates for the console, to enable inventory and filebucket viewing.
- Edit `passenger-extra.conf`

- Restart the `pe-httpd` service.

You can upgrade agent nodes after upgrading the puppet master and console. After upgrading an agent node, you must:

- Manually edit `puppet.conf`.

Stop `pe-httpd`

For the duration of these manual steps, Puppet Enterprise's web server should be stopped.

```
$ sudo /etc/init.d/pe-httpd stop
```

Create a New Inventory Database

To support the inventory service, you must manually create a new database for puppet master to store node facts in. To do this, use the `mysql` client on the node running the database server. (This will almost always be the same server running the puppet master and console.)

```
# mysql -uroot -p
Enter password:
mysql> CREATE DATABASE console_inventory_service;
mysql> GRANT ALL PRIVILEGES ON console_inventory_service.* TO
'<USER>'@'localhost';
```

Replace `<USER>` with the MySQL user name you gave Dashboard during your original installation.

Edit Puppet Master's `/etc/puppetlabs/puppet/puppet.conf`

- To support the inventory service, you must configure Puppet to save facts to a MySQL database.

  ```
  [master]
      # ...
      facts_terminus = inventory_active_record
      dbadapter = mysql
      dbname = console_inventory_service
      dbuser = <CONSOLE/DASHBOARD'S MYSQL USER>
      dbpassword = <PASSWORD FOR CONSOLE'S MYSQL USER>
      dbserver = localhost
  ```

  If you chose a different MySQL user name for Puppet Dashboard when you originally installed PE, use that user name as the `dbuser` instead of "dashboard". If the database is served by a remote machine, use that server's hostname instead of "localhost".

- If you configured the puppet master to not send reports to the Dashboard, you must configure it to report to the console now:

```
[master]
    # ...
    reports = https, store
    reporturl = https://<CONSOLE HOSTNAME>:<PORT>/reports/upload
```

- Puppet agent on this node also has some new requirements:

```
[agent]
    # support filebucket viewing when using compliance features:
    archive_files = true
    # if you didn't originally enable pluginsync, enable it now:
    pluginsync = true
```

Edit Puppet Master's `/etc/puppetlabs/puppet/auth.conf`

To support the inventory service, you must add the following two stanzas to your puppet master's `auth.conf` file:

```
# Allow the console to retrieve inventory facts:

path /facts
auth yes
method find, search
allow pe-internal-dashboard

# Allow puppet master to save facts to the inventory:

path /facts
auth yes
method save
allow <PUPPET MASTER'S CERTNAME>
```

These stanzas must be inserted before the final stanza, which looks like this:

```
path /
auth any
```

If you paste the new stanzas after this final stanza, they will not take effect.

Edit `/etc/puppetlabs/puppet/manifests/site.pp`

You must add the following lines to site.pp in order to view file contents in the console:

```
# specify remote filebucket
filebucket { 'main':
  server => '<puppet master's hostname>',
  path => false,
}
```

```
File { backup => 'main' }
```

Edit `/etc/puppetlabs/puppet-dashboard/settings.yml`

Change the following three settings to point to one of the puppet master's valid DNS names:

```
ca_server: '<PUPPET MASTER HOSTNAME>'
inventory_server: '<PUPPET MASTER HOSTNAME>'
file_bucket_server: '<PUPPET MASTER HOSTNAME>'
```

Change the following two settings to true:

```
enable_inventory_service: true
use_file_bucket_diffs: true
```

Ensure that the following settings exist and are set to the suggested values; if any are missing, you will need to add them to settings.yml yourself:

```
private_key_path: 'certs/pe-internal-dashboard.private_key.pem'
public_key_path: 'certs/pe-internal-dashboard.public_key.pem'
ca_crl_path: 'certs/pe-internal-dashboard.ca_crl.pem'
ca_certificate_path: 'certs/pe-internal-dashboard.ca_cert.pem'
certificate_path: 'certs/pe-internal-dashboard.cert.pem'
key_length: 1024
cn_name: 'pe-internal-dashboard'
```

Generate and Sign Console Certificates

First, navigate to the console's installation directory:

```
$ cd /opt/puppet/share/puppet-dashboard
```

Next, start a temporary WEBrick puppet master:

```
$ sudo /opt/puppet/bin/puppet master
```

Next, create a keypair and request a certificate:

```
$ sudo /opt/puppet/bin/rake cert:create_key_pair
$ sudo /opt/puppet/bin/rake cert:request
```

Next, sign the certificate request:

```
$ sudo /opt/puppet/bin/puppet cert sign dashboard
```

Next, retrieve the signed certificate:

```
$ sudo /opt/puppet/bin/rake cert:retrieve
```

Next, stop the temporary puppet master:

```
$ sudo kill $(cat $(puppet master --configprint pidfile) )
```

Finally, chown the certificates directory to `puppet-dashboard`:

```
$ sudo chown -R puppet-dashboard:puppet-dashboard certs
```

TROUBLESHOOTING

If these rake tasks fail with errors like `can't convert nil into String`, you may be missing a certificate-related setting from the settings.yml file. Go back to the previous section and make sure all of the required settings exist.

Start `pe-httpd`

You can now start PE's web server again.

```
$ sudo /etc/init.d/pe-httpd start
```

Edit `puppet.conf` on Each Agent Node

On each agent node you upgrade to PE 2.0, make the following edits to `/etc/puppetlabs/puppet/puppet.conf`:

```
[agent]
    # support filebucket viewing when using compliance features:
    archive_files = true
    # if you didn't originally enable pluginsync, enable it now:
    pluginsync = true
```

← Installing: Basic Installation — Index — Installing: Uninstalling →

PE 2.0 » Installing » Uninstalling

Uninstalling Puppet Enterprise

Use the `puppet-enterprise-uninstaller` script to uninstall PE. This script can remove a working PE installation, or undo a partially failed installation to prepare for a re-install.

Note For PE 2.0.0

The uninstaller script was not included in the PE 2.0.0 tarball. If the tarball you downloaded does not have the uninstaller, you must first download it:

- Click here to download the uninstaller, or use `curl` or `wget` to download it directly to the target machine.
- Copy the uninstaller to the target machine, and move it into the directory which contains the installer script. The uninstaller and the installer must be in the same directory.
- Make the uninstaller executable, then run it:

```
# sudo chmod +x puppet-enterprise-uninstaller
# sudo ./puppet-enterprise-uninstaller
```

Using the Uninstaller

To run the uninstaller, ensure that it is in the same directory as the installer script, and run it with root privileges from the command line:

```
# sudo ./puppet-enterprise-uninstaller
```

The uninstaller will ask you to confirm that you want to uninstall.

By default, the uninstaller will remove the Puppet Enterprise software, users, logs, cron jobs, and caches, but it will leave your modules, manifests, certificates, databases, and configuration files in place, as well as the home directories of any users it removes.

You can use the following command-line flags to change the installer's behavior:

Uninstaller Options

`-p`
Purge additional files; with this flag, the uninstaller will also remove all configuration files, modules, manifests, certificates, and the home directories of any users created by the PE installer.
`-d`
Also remove any databases during the uninstall.
`-h`
Display a help message.
`-n`
Run in 'noop' mode; show commands that would have been run during installation without running them.

`-y`
Don't ask to confirm uninstallation, assuming an answer of yes.

`-s`
Save an answer file and quit without uninstalling.

`-a`
Read answers from file and fail if an answer is missing. See the uninstaller answers section of the answer file reference for a list of available answers.

`-A`
Read answers from file and prompt for input if an answer is missing. See the uninstaller answers section of the answer file reference for a list of available answers.

Thus, to remove every trace of PE from a system, you would run:

```
# sudo ./puppet-enterprise-uninstaller -d -p
```

← Installing: Upgrading — Index — Installing: Automated Installation →

PE 2.0 » Installing » Automated Installation

← Installing: Uninstalling — Index — Installing: Answer File Reference →

Automated Installation

To streamline deployment, the PE installer can run non-interactively. To do this, you must:

- Create an answer file
- Run the installer with the `-a` or `-A` flags

Instead of interviewing a user to customize the installation, the installer will read your choices from the answer file and act on them immediately.

Automated installation can greatly speed up large deployments, and is crucial when installing PE with the cloud provisioning tools.

Obtaining an Answer File

Answer files are simply shell scripts that declare variables used by the installer:

```
q_install=y
q_puppet_cloud_install=n
q_puppet_enterpriseconsole_install=n
q_puppet_symlinks_install=y
q_puppetagent_certname=webmirror1.example.com
q_puppetagent_install=y
q_puppetagent_server=puppet
q_puppetmaster_install=n
q_vendor_packages_install=y
```

(A full list of these variables is available in the answer file reference.)

To obtain an answer file, you can:

- Use one of the example files provided in the installer's `answers` directory
- Retrieve the `answers.lastrun` file from a node on which you've already installed PE
- Run the installer with the `-s <FILE>` flag, which saves an answer file without installing
- Write one by hand

You must hand-edit any pre-made answer file before using it, as new nodes will need, at a minimum, a unique agent certname.

Editing Answer Files

Although you can use literal strings in an answer file for one-off installations, you should fill certain variables dynamically with bash subshells if you want your answer files to be reusable.

To run a subshell that will return the output of its command, use either the `$()` notation...

```
q_puppetagent_certname=$(hostname -f)
```

…or backticks:

```
q_puppetagent_certname=`uuidgen`
```

Answer files can also contain arbitrary shell code and control logic, but you will probably be able to get by with a few simple name-discovery commands.

See the answer file reference for a complete list of variables and the conditions where they're needed, or simply start editing one of the example files in `answers/`.

Running the Installer in Automated Mode

Once you have your answer file, simply run the installer with the `-a` or `-A` option, providing your answer file as an argument:

```
# sudo ./puppet-enterprise-installer -a ~/normal_agent.answers
```

- Installing with the `-a` option will fail if any required variables are not set.
- Installing with the `-A` option will prompt the user for any missing answers.

← Installing: Uninstalling — Index — Installing: Answer File Reference →

PE 2.0 » Installing » Answer File Reference

← Installing: Automated Installation — Index — Installing: What gets installed where? →

Answer File Reference

Answer files are used when doing an automated installation of PE. See the chapter on automated installation for more details.

Answer File Syntax

Answer files consist of normal shell script variable assignments:

```
q_puppet_enterpriseconsole_database_port=3306
```

Boolean answers should use Y or N (case-insensitive) rather than true, false, 1, or 0.

A variable can be omitted if another answer ensures that it won't be used (i.e. `q_puppetmaster_certname` can be left blank if `q_puppetmaster_install` = n).

Answer files can include arbitrary bash control logic, and can assign variables with commands in subshells (`$(command)`). For example, to set an agent node's certname to its fqdn:

```
q_puppetagent_certname=$(hostname -f)
```

To set it to a UUID:

```
q_puppetagent_certname=$(uuidgen)
```

Sample Answer Files

PE includes a collection of sample answer files in the `answers` directory of your distribution tarball. A successful installation will also save an answer file called `answers.lastrun`, which can be used as a foundation for later installations. Finally, you can generate a new answer file without installing by running the installer with the `-s` option and a filename argument.

Installer Answers

Global Answers

These answers are always needed.

`q_install`
Y or N — Whether to install. Answer files must set this to Y.
`q_vendor_packages_install`
Y or N — Whether the installer has permission to install additional packages from the OS's repositories. If this is set to N, the installation will fail if the installer detects missing dependencies.

`q_puppet_symlinks_install`
Y or N — Whether to make the Puppet tools more visible to all users by installing symlinks in `/usr/local/bin`.

Roles

These answers are always needed.

`q_puppetmaster_install`
Y or N — Whether to install the puppet master role.
`q_puppet_enterpriseconsole_install`
Y or N — Whether to install the console role.
`q_puppetagent_install`
Y or N — Whether to install the puppet agent role.
`q_puppet_cloud_install`
Y or N — Whether to install the cloud provisioner role.

Puppet Agent Answers

These answers are always needed.

`q_puppetagent_certname`
String — An identifying string for this agent node. This per-node ID must be unique across your entire site. Fully qualified domain names are often used as agent certnames.
`q_puppetagent_server`
String — The hostname of the puppet master server. For the agent to trust the master's certificate, this must be one of the valid DNS names you chose when installing the puppet master.

Puppet Master Answers

These answers are only needed if you are installing the puppet master role.

`q_puppetmaster_certname`
String — An identifying string for the puppet master. This ID must be unique across your entire site. The server's fully qualified domain name is often used as the puppet master's certname.
`q_puppetmaster_dnsaltnames`
String — Valid DNS names at which the puppet master can be reached. Must be a comma-separated list. In a normal installation, defaults to `<hostname>,<hostname.domain>,puppet,puppet.<domain>`.
`q_puppetmaster_enterpriseconsole_hostname`
String — The hostname of the server running the console role. Only needed if you are not installing the console role on the puppet master server.
`q_puppetmaster_enterpriseconsole_port`
Integer — The port on which to contact the console server. Only needed if you are not installing the console role on the puppet master server.

Console Answers

These answers are only needed if you are installing the console role.

`q_puppet_enterpriseconsole_httpd_port`
Integer — The port on which to serve the console. If this is set to 443, you can access the console from a web browser without manually specifying a port.
`q_puppet_enterpriseconsole_auth_user`
String — The user name for accessing the console's web interface.
`q_puppet_enterpriseconsole_auth_password`
String — The password for accessing the console's web interface. Must be longer than four characters.
`q_puppet_enterpriseconsole_inventory_certname`
String — An identifying string for the inventory service. This ID must be unique across your

String — Particularly long string for the inventory service. This ID must be unique across your entire site. Only needed if you are not installing the puppet master role on the console server.

`q_puppet_enterpriseconsole_inventory_dnsaltnames`

String — Valid DNS names at which the console server can be reached. Only needed if you are not installing the puppet master role on the console server. Must be a comma-separated list. In a normal installation, defaults to `<hostname>,<hostname.domain>,puppetinventory,puppetinventory.<domain>`.

`q_puppet_enterpriseconsole_database_install`

Y or N — Whether to install and configure a new MySQL database from the OS's package repositories. If set to Y, the installer will also create a new database and user with the `..._name`, `..._user`, and `..._password` answers below.

`q_puppet_enterpriseconsole_setup_db`

Y or N — Whether to automatically configure the console and inventory service databases. Only used when `q_puppet_enterpriseconsole_database_install` is N.

`q_puppet_enterpriseconsole_database_root_password`

String — The password for MySQL's root user. When `q_puppet_enterpriseconsole_database_install` is Y, this will set the root user's password; when `q_puppet_enterpriseconsole_setup_db` is Y, it will be used to log in and automatically configure the necessary databases. If neither of these answers is Y, this answer is not used.

`q_puppet_enterpriseconsole_database_remote`

Y or N — Whether the pre-existing database is on a remote MySQL server. Only used when `q_puppet_enterpriseconsole_database_install` is N.

`q_puppet_enterpriseconsole_database_host`

String — The hostname of the remote MySQL server. Only used when `q_puppet_enterpriseconsole_database_remote` is Y.

`q_puppet_enterpriseconsole_database_port`

String — The port used by the remote MySQL server. Only used when `q_puppet_enterpriseconsole_database_remote` is Y. In a normal installation, defaults to `3306`.

`q_puppet_enterpriseconsole_database_name`

String — The database the console will use. Note that if you are not automatically configuring the databases, this database must already exist on the MySQL server.

`q_puppet_enterpriseconsole_database_user`

String — The MySQL user the console will use. Note that if you are not automatically configuring the databases, this user must already exist on the MySQL server and must be able to edit the console's database.

`q_puppet_enterpriseconsole_database_password`

String — The password for the console's MySQL user.

Upgrader Answers

`q_upgrade_installation`

Y or N — Whether to upgrade. Answer files must set this to Y.

`q_puppet_cloud_install`

Y or N — Whether to install the cloud provisioner tools on this node during the upgrade. Previous versions of PE did not include the cloud provisioner tools.

`q_puppet_enterpriseconsole_auth_user`

String — A user name for accessing the console. Previous versions of Puppet Enterprise did not secure the Dashboard with a username and password. Only required if this node has the console role (previously Puppet Dashboard) installed.

`q_puppet_enterpriseconsole_auth_password`

String — A password for accessing the console. Previous versions of Puppet Enterprise did not secure the Dashboard with a username and password. Only required if this node has the console role (previously Puppet Dashboard) installed.

`q_vendor_packages_install`

Y or N — Whether to install additional packages from your OS vendor's repository, if the upgrader determines any are needed.

`q_upgrade_remove_mco_homedir`

Y or N — Whether to delete the mco user's home directory. (The mco user from PE 1.2 was replaced with the peadmin user in PE 2.0.)

`q_upgrade_install_wrapper_modules`

Y or N — Whether to install wrapper modules, so that you can continue to use the Puppet modules provided with PE 1.2 under their previous names as well as their new names. (The `accounts`, `baselines`, and `mcollectivepe` modules from PE 1.2 were renamed to

`pe_accounts`, `pe_compliance`, and `pe_mcollective` in PE 2.0.)

Uninstaller Answers

`q_pe_uninstall`
Y or N — Whether to uninstall. Answer files must set this to Y.

`q_pe_purge`
Y or N — Whether to purge additional files when uninstalling, including all configuration files, modules, manifests, certificates, and the home directories of any users created by the PE installer.

`q_pe_remove_db`
Y or N — Whether to remove any PE-specific databases when uninstalling.

`q_pe_db_root_pass`
String — The MySQL root user's password, to be used when deleting databases. Only used when `q_pe_remove_db` is Y.

← Installing: Automated Installation — Index — Installing: What gets installed where? →

PE 2.0 » Installing » What Gets Installed Where?

← Installing: Answer File Reference — Index — Console: Accessing the Console →

What Gets Installed Where?

License File

Your PE license file (which was emailed to you when you purchased Puppet Enterprise) should be placed at `/etc/puppetlabs/license.key`.

Puppet Enterprise can be evaluated with a complementary ten-node license; beyond that, a commercial per-node license is required for use. A license key file will have been emailed to you after your purchase, and the puppet master will look for this key at `/etc/puppetlabs/license.key`. Puppet will log warnings if the license is expired or exceeded, and you can view the status of your license by running `puppet license` at the command line on the puppet master.

To purchase a license, please see the Puppet Enterprise pricing page, or contact Puppet Labs at sales@puppetlabs.com or (877) 575-9775. For more information on licensing terms, please see the licensing FAQ. If you have misplaced or never received your license key, please contact sales@puppetlabs.com.

Configuration Files

Puppet Enterprise's configuration files all live under `/etc/puppetlabs`, with subdirectories for each of PE's components.

- Puppet's `confdir` is in `/etc/puppetlabs/puppet`. This directory contains the `puppet.conf` file, the site manifest (`manifests/site.pp`), and the `modules` directory.
- MCollective's config files are in `/etc/puppetlabs/mcollective`.
- The console's config files are in `/etc/puppetlabs/puppet-dashboard`.

Documentation

Man pages for the Puppet subcommands are generated on the fly. To view them, run `puppet man <SUBCOMMAND>`.

The `pe-man` command from previous versions of Puppet Enterprise is still functional, but it is deprecated and is slated for removal in a future release.

Software

All PE software is installed under `/opt/puppet`.

- Executable binaries are in `/opt/puppet/bin` and `/opt/puppet/sbin`.
- Optionally, you can choose at install time to symlink the most common binaries into

`/usr/local/bin`.

- The Puppet modules included with PE are installed in `/opt/puppet/share/puppet/modules`. Don't edit this directory to add modules of your own; instead, install them in `/etc/puppetlabs/puppet/modules`.
- MCollective plugins are installed in `/opt/puppet/libexec/mcollective/`. If you are adding new plugins to your PE agent nodes, you should distribute them via Puppet.

Services

PE uses the following services:

- `pe-puppet` (on EL platforms) and `pe-puppet-agent` (on Debian-based platforms) — The puppet agent daemon. Runs on every agent node.
- `pe-httpd` — Apache 2, which manages and serves puppet master and the console on servers with those roles. (Note that PE uses Passenger to run puppet master, instead of running it as a standalone daemon.)
- `pe-mcollective` — The MCollective server. Runs on every agent node.
- `pe-puppet-dashboard-workers` — A supervisor that manages the console's background processes. Runs on servers with the console role.
- `pe-activemq` — The ActiveMQ message server, which passes messages to the MCollective servers on agent nodes. Runs on servers with the puppet master role.

User Accounts

PE creates the following users:

- `peadmin` — An administrative account which can issue MCollective client commands. This is the only PE user account intended for use in a login shell. See the chapter on orchestration for more about this user. This user exists on servers with the puppet master role, and replaces the `mco` user that was present in PE 1.2.
- `pe-puppet` — A system user which runs the puppet master processes spawned by Passenger.
- `pe-apache` — A system user which runs Apache (`pe-httpd`).
- `pe-activemq` — A system user which runs the ActiveMQ message bus used by MCollective.
- `puppet-dashboard` — A system user which runs the console processes spawned by Passenger.

Log Files

The software distributed with Puppet Enterprise generates the following log files:□

Puppet Master

- `/var/log/pe-httpd/access.log`

- `/var/log/pe-httpd/error.log`

These logs are solely for HTTP activity; the puppet master service logs most of its activity to the syslog service. Your syslog configuration dictates where these messages will be saved, but the default location is `/var/log/messages` on Linux and `/var/adm/messages` on Solaris.

Puppet Agent

The puppet agent service logs its activity to the syslog service. Your syslog configuration dictates where these messages will be saved, but the default location is `/var/log/messages` on Linux and `/var/adm/messages` on Solaris.

ActiveMQ

- `/var/log/pe-activemq/wrapper.log`
- `/var/log/pe-activemq/activemq.log`
- `/var/opt/puppet/activemq/data/kahadb/db-1.log`
- `/var/opt/puppet/activemq/data/audit.log`

Orchestration Service

This log is maintained by the orchestration service, which is installed on all nodes.

- `/var/log/pe-mcollective/mcollective.log`

Console

- `/var/log/pe-httpd/puppetdashboard.access.log`
- `/var/log/pe-httpd/puppetdashboard.error.log`
- `/var/log/pe-puppet-dashboard/delayed_job.log`
- `/var/log/pe-puppet-dashboard/mcollective_client.log`
- `/var/log/pe-puppet-dashboard/production.log`

Miscellaneous

These files may or may not be present.

- `/var/log/pe-httpd/other_vhosts_access.log`
- `/var/log/pe-puppet/masterhttp.log`
- `/var/log/pe-puppet/rails.log`

← Installing: Answer File Reference — Index — Console: Accessing the Console →

PE 2.0 » Console » Accessing

Accessing the Console

The console is Puppet Enterprise's web GUI. Use it to:

- Browse and edit resources on your nodes
- Trigger Puppet runs at will
- View reports and activity graphs
- Assign Puppet classes to nodes
- View inventory data
- Track compliance audits
- Invoke MCollective agents on your nodes

Browser Requirements

Puppet Labs supports the following browsers for use with the console:

- Chrome (current versions)
- Firefox 3.5 and higher
- Safari 4 and higher
- Internet Explorer 9 and higher

Although we plan to fully support Internet Explorer 8 in the near future, it currently stalls indefinitely when trying to load the console's live management page. Other browsers may or may not work, and have not been intensively tested with the console.

Reaching the Console

The console will be served as a website over SSL, on whichever port you chose when installing the console role.

Let's say your console server is `console.domain.com`. If you chose to use the default port of 443, you can omit the port from the URL and would reach the console by navigating to:

```
https://console.domain.com
```

If you chose to use port 3000, you would reach it at:

```
https://console.domain.com:3000
```

Note the `https` protocol handler — you cannot reach the console over plain `http`.

Accepting the Console's Certificate

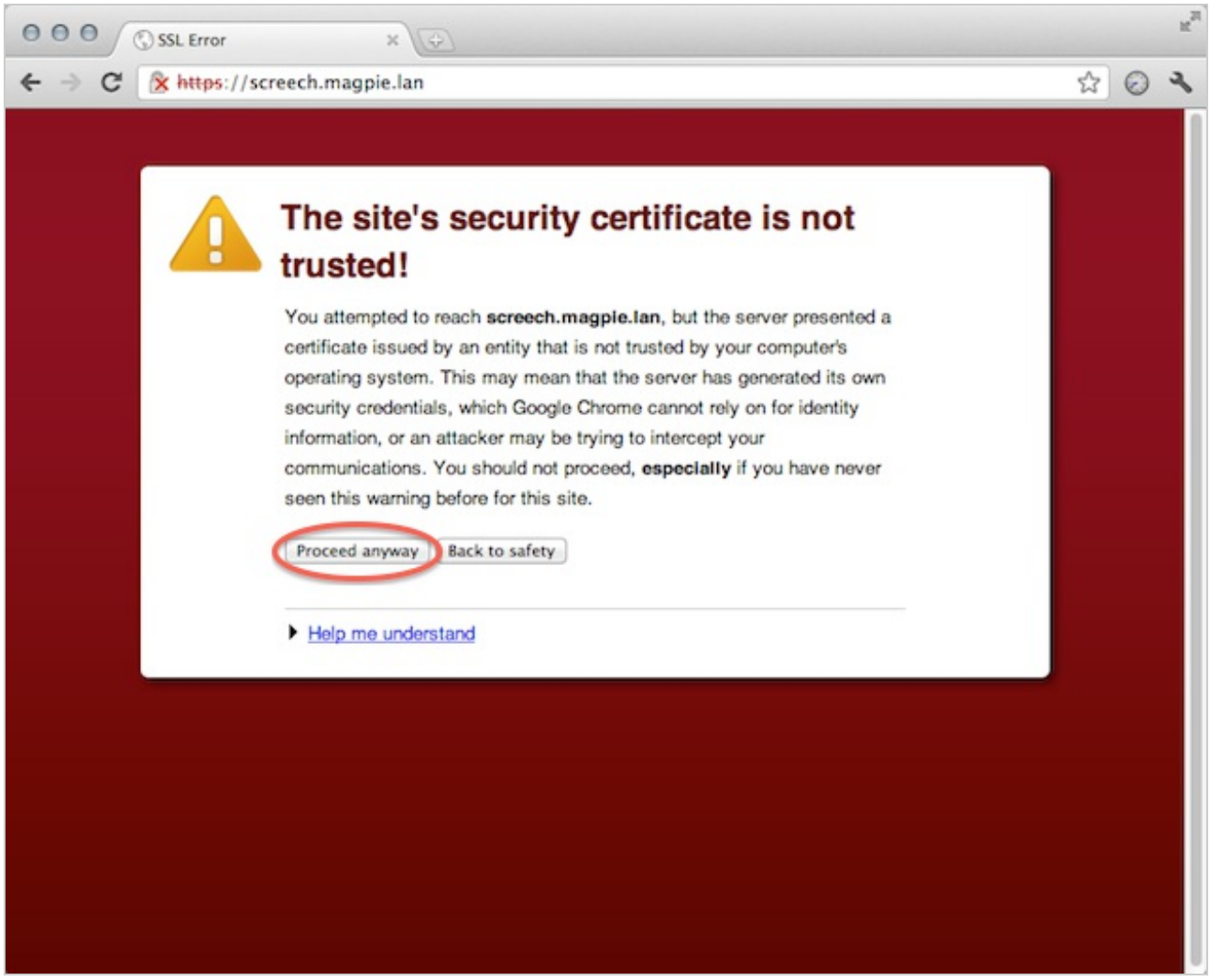
The console uses an SSL certificate created by your own local Puppet certificate authority. Since this

authority is specific to your site, web browsers won't know it or trust it, and you'll have to add a security exception in order to access the console.

This is safe to do. Your web browser will warn you that the console's identity hasn't been verified by one of the external authorities it knows of, but that doesn't mean it's untrustworthy: since you or another administrator at your site is in full control of which certificates the Puppet certificate authority signs, the authority verifying the site is you.

Accepting the Certificate in Google Chrome or Chromium

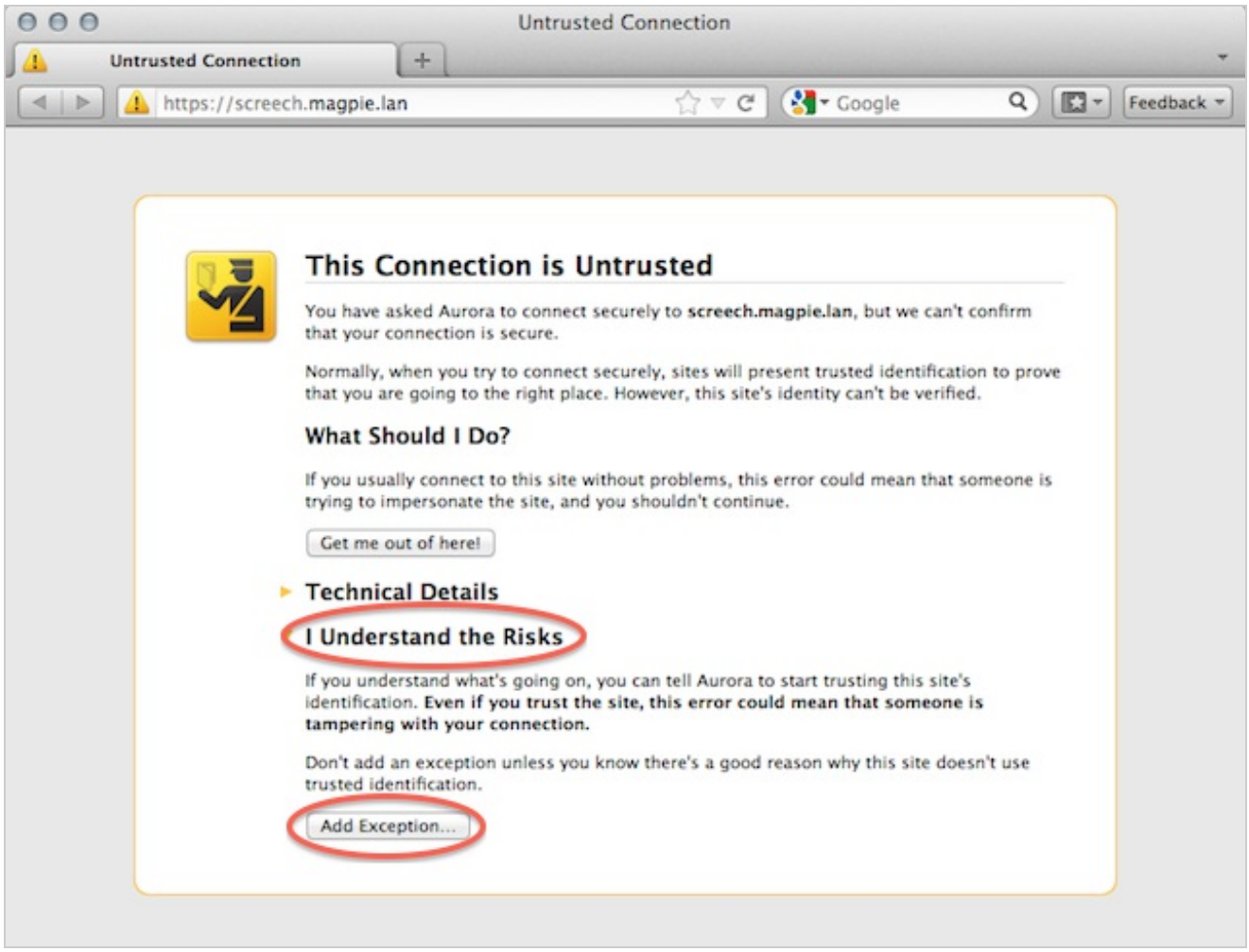
Use the "Proceed anyway" button on Chrome's warning page.



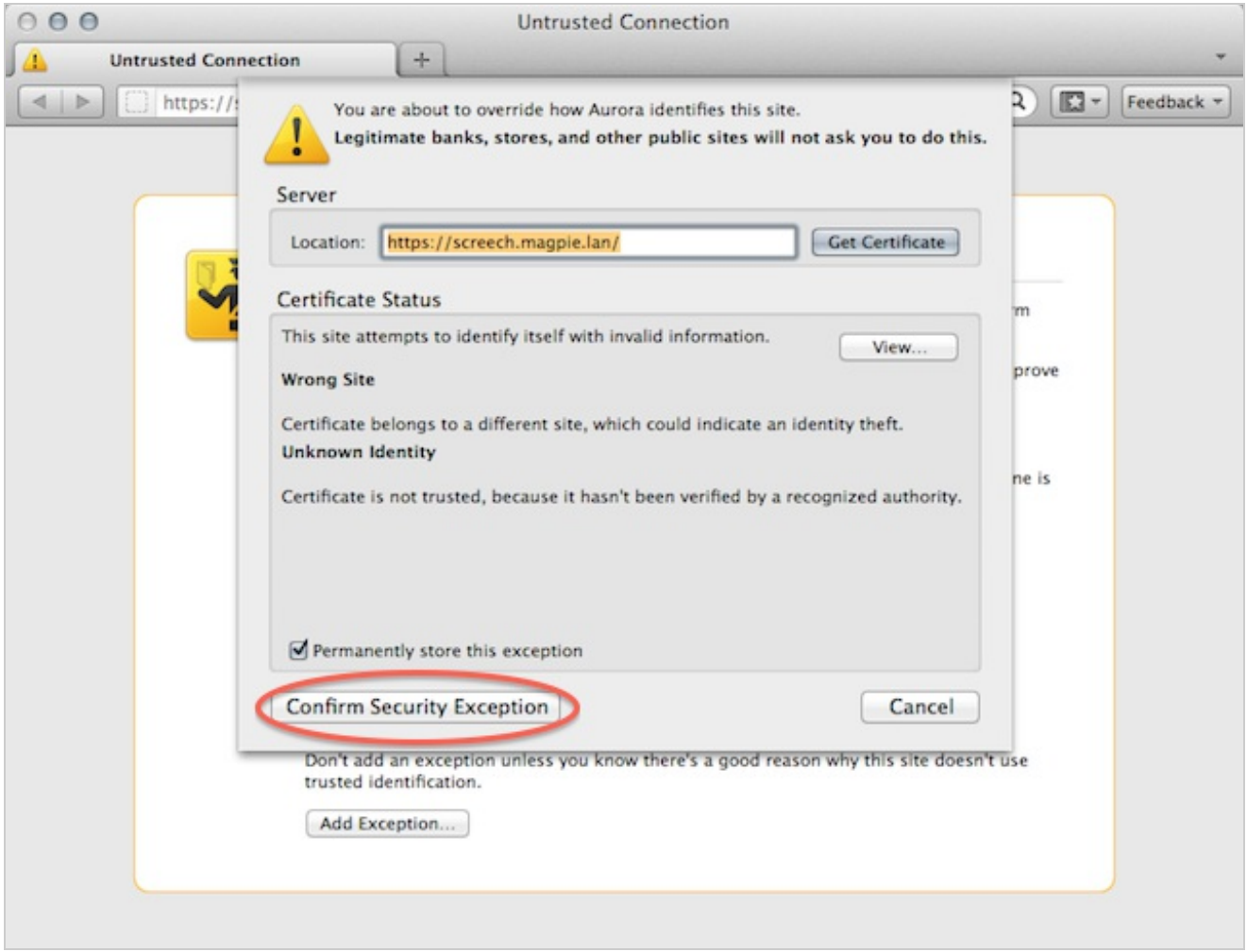
Accepting the Certificate in Mozilla Firefox

Click "I Understand the Risks" to reveal more of the page, then click the "Add Exception…" button. On the dialog this raises, click the "Confirm Security Exception" button.

Step 1:

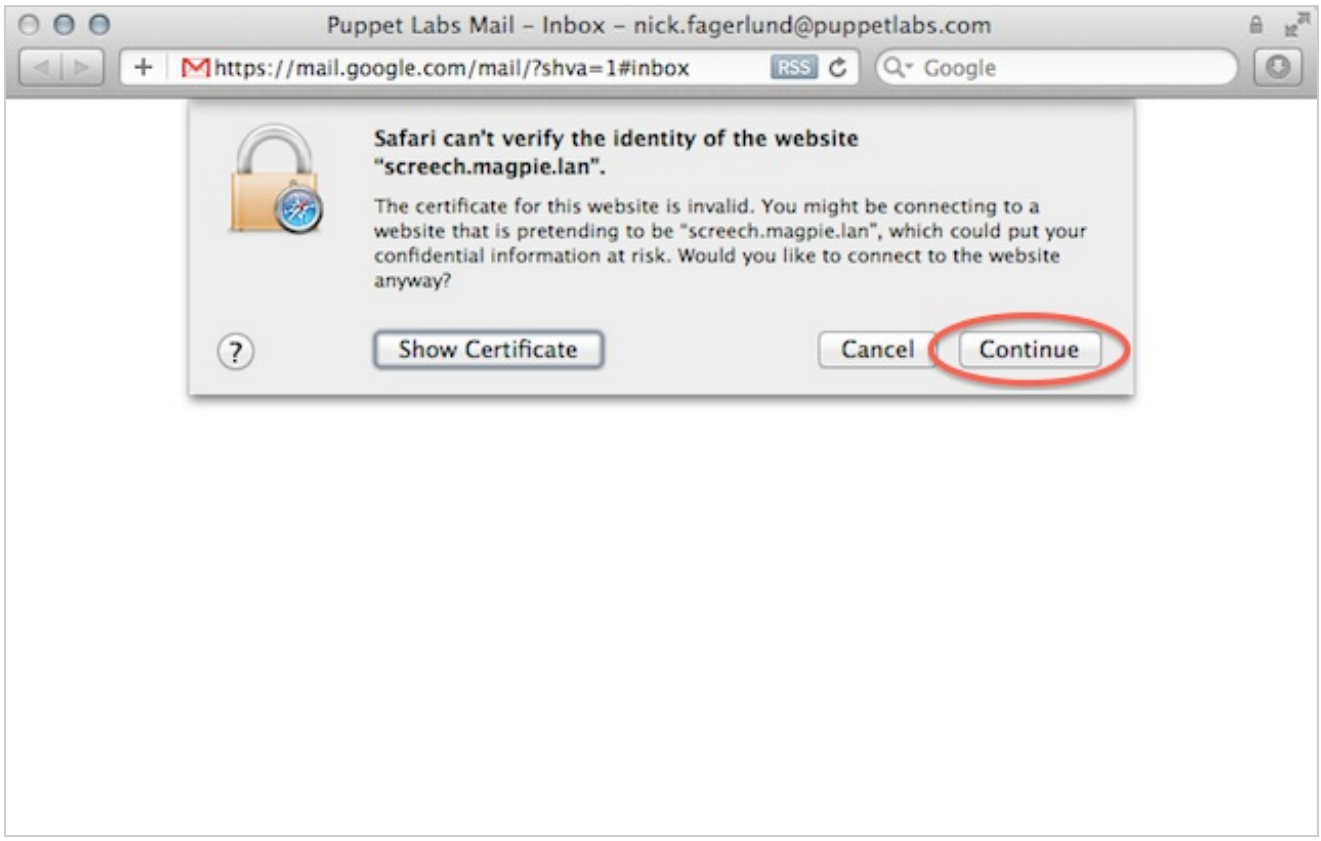

Untrusted Connection
Untrusted Connection
https://screech.magpie.lan
Google
Feedback
This Connection is Untrusted
You have asked Aurora to connect securely to screech.magpie.lan, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.
What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.
Get me out of here!
Technical Details
I Understand the Risks
If you understand what's going on, you can tell Aurora to start trusting this site's identification. Even if you trust the site, this error could mean that someone is tampering with your connection.
Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.
Add Exception...

Step 2:



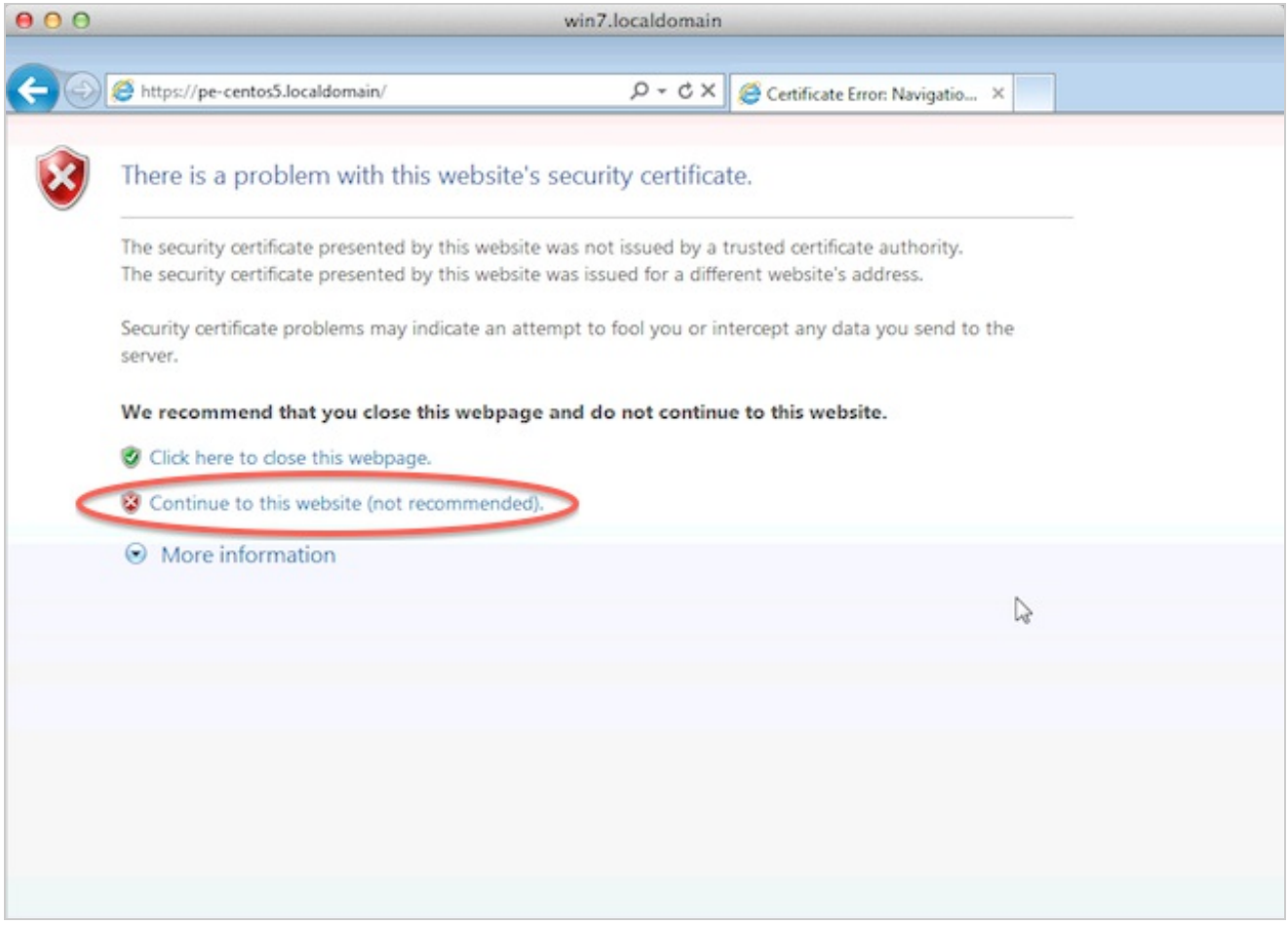
Accepting the Certificate in Apple Safari

Click the “Continue” button on the warning dialog.



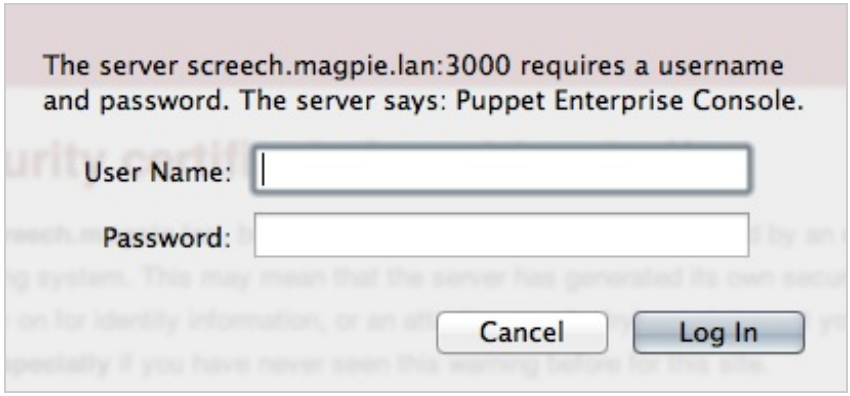
Accepting the Certificate in Microsoft Internet Explorer

Click the "Continue to this website (not recommended)" link on the warning page.



Logging In

For security, the console requires a user name and password for access. Use the user and password you chose when you installed the console role, or get credentials from your site's lead administrator.



Since the console is the main point of control for your infrastructure, you will probably want to decline your browser's offer to remember its password.□

← Installing: What gets installed where? — Index — Console: Navigating the Console →

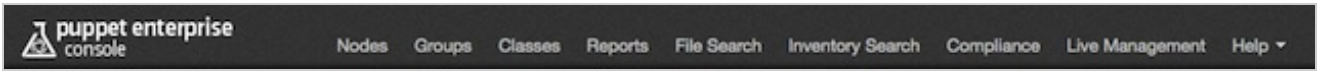
PE 2.0 » Console » Navigating

← Console: Accessing the Console — Index — Console: Viewing Reports and Inventory Data →

Navigating the Console

Getting Around

Navigate between sections of the console using the navigation bar at the top.

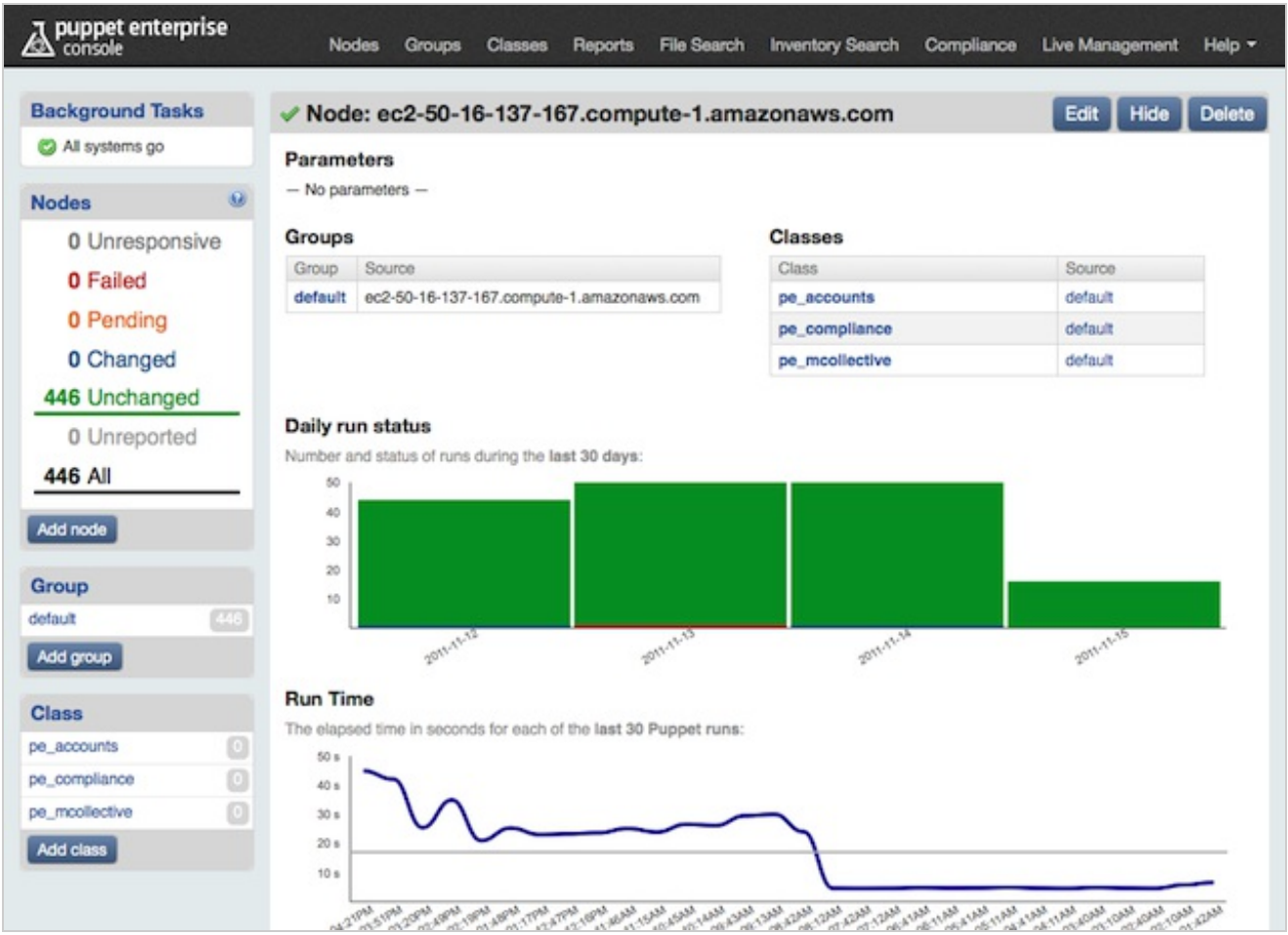


What's in the Console?

The console deals with three main objects:

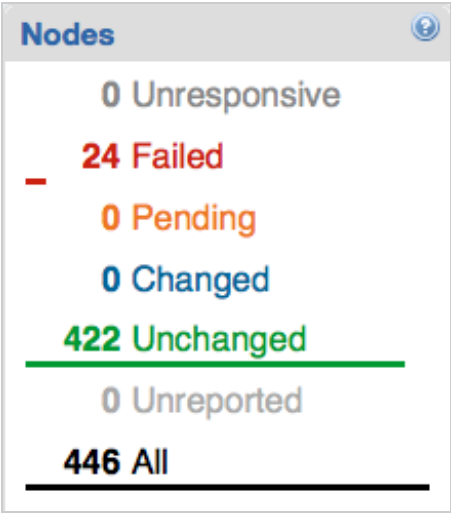
- A node represents a single system being managed by Puppet. It can have classes applied to it, and can be a member of groups.
- A group is an arbitrary set of nodes. It can have classes applied to it, and can contain nodes.
- A class represents a Puppet class available in your puppet master's collection of modules. It can be applied to nodes or to groups.

Nodes



After a node completes its first Puppet run (which may take up to 30 minutes after you've signed its certificate), it will appear in the console, and can be added to groups and have classes applied to it.

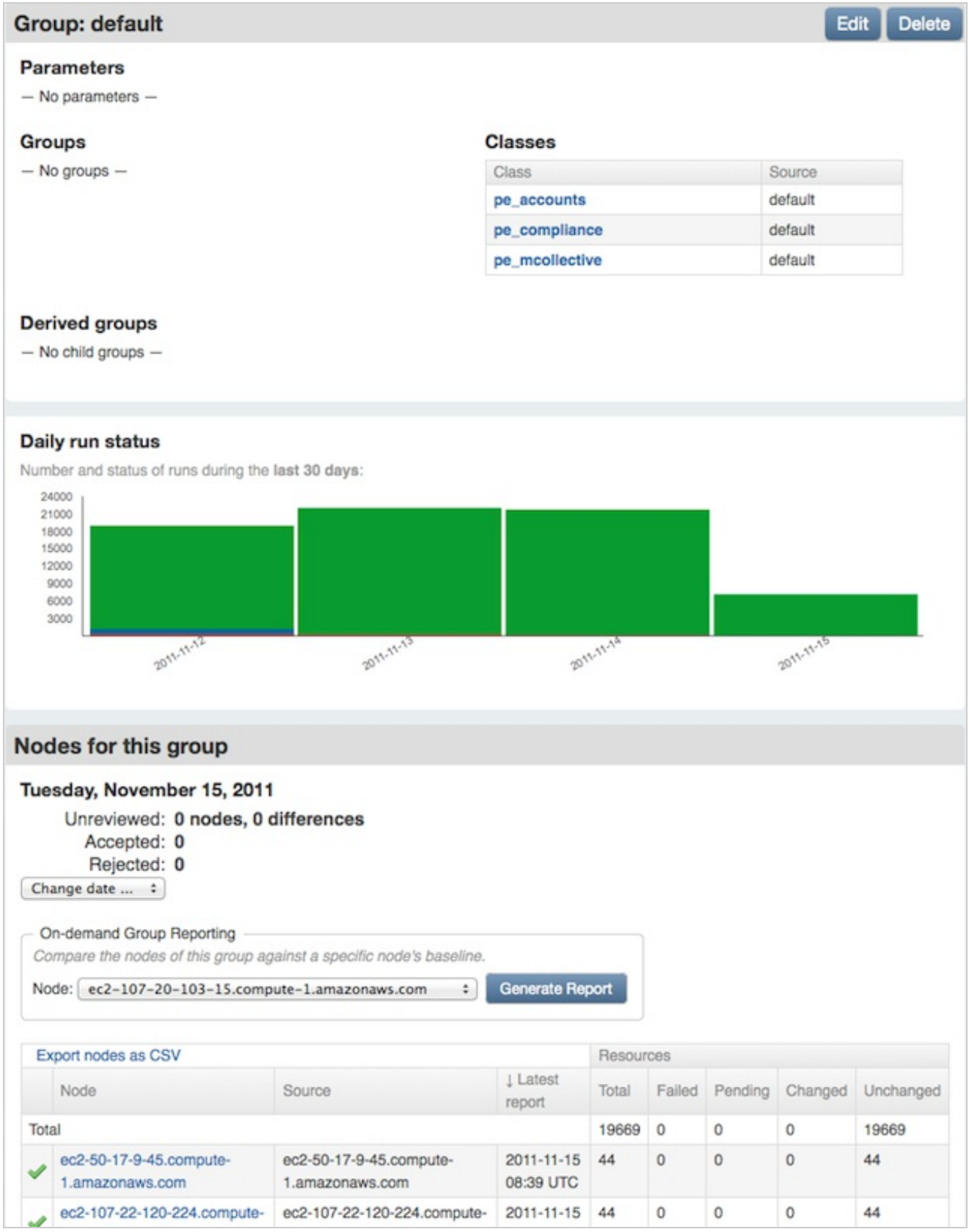
Since the console receives a report every time Puppet runs on a node, it keeps a running count in the sidebar of what state your nodes are in:



This can tell you at a glance whether your nodes have suddenly started failing their Puppet runs, whether any nodes have stopped responding, and whether Puppet is making many changes to your systems. You can click through these state totals to see complete lists of nodes in each state.

Individual node pages contain graphs of recent runs, lists of reports, inventory data, compliance data, and any classes the node has or groups it's a part of.

Groups



Groups contain nodes. Any classes applied to a group will also be applied to all the nodes in it.

Classes

Classes aren't automatically detected or validated; you have to enter a class's name yourself before you can apply it to a node or group. Once you do, though, you're good to go; Puppet will apply it as needed, and you can click the class in the console for a view of which nodes it's been assigned to.

PE 2.0 » Console » Viewing Reports

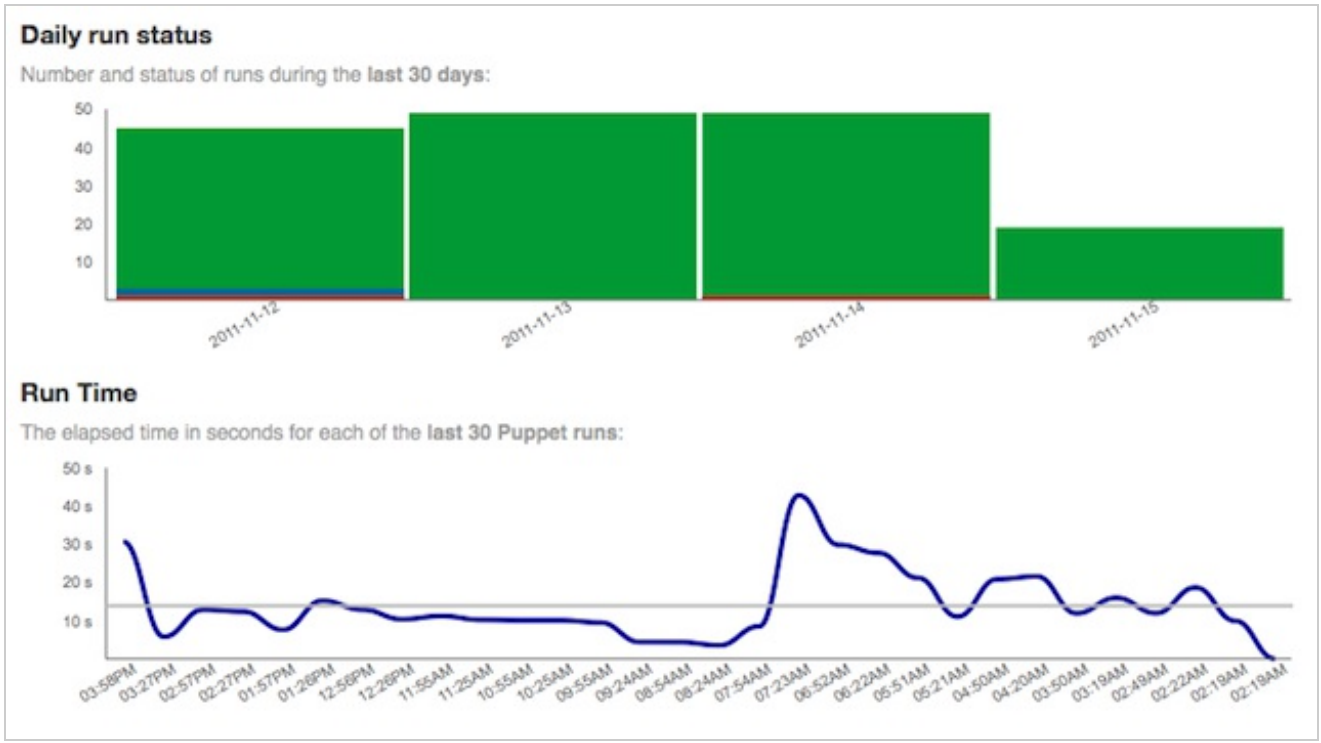
Viewing Reports and Inventory Data

When nodes fetch their configurations from the puppet master, they send back inventory data and a report of their run. These end up in the console, where you can view them in that node's page.

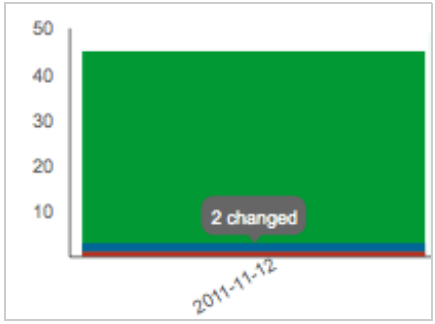
Reading Reports

Graphs

Each node page has a pair of graphs: a histogram showing the number of runs per day and the results of those runs, and a line chart tracking how long each run took.

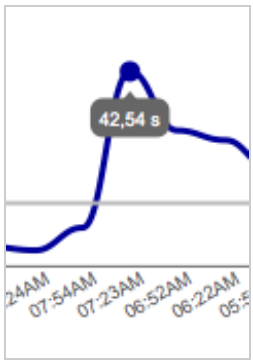


The daily run status histogram is broken down with the same colors that indicate run statuses in the console's sidebar: red for failed runs, orange for pending runs (where a change would have been made, but the resource to be changed was marked as no-op), blue for successful runs where changes were made, and green for successful runs that did nothing. You can hover over a block of color for a tooltip showing how many runs of that type occurred:



Note: Run status histograms also appear on group pages, class pages, and run status pages.

The run time chart graphs how long each of the last 30 Puppet runs took to complete. A longer run usually means changes were made, but could also indicate heavy server load or some other circumstance. You can hover over a point on the line for a tooltip showing the number of seconds it represents:



Normal Reports

Each node page has a short list of recent reports, with a “more” button at the bottom for viewing older reports:

Recent reports (162)

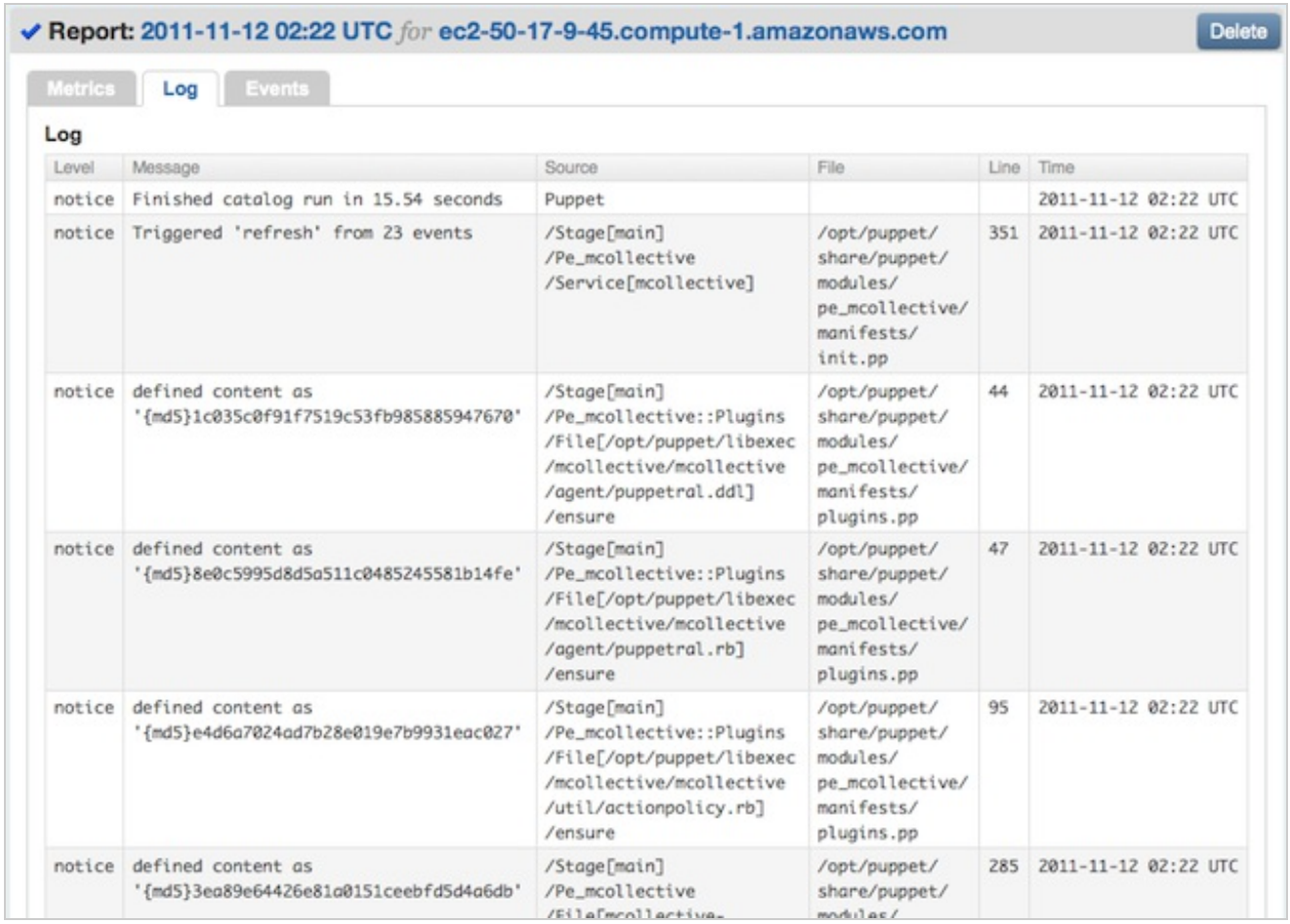
	Reported at ↓	Total	Failed	Changed	Unchanged	Pending	Skipped	Failed restarts	Config retrieval	Runtime
✓	2011-11-15 10:11 UTC	44	0	0	44	0	6	0	10.28 s	33.93 s
✓	2011-11-15 09:40 UTC	44	0	0	44	0	6	0	12.96 s	49.13 s
✓	2011-11-15 09:10 UTC	44	0	0	44	0	6	0	3.69 s	37.96 s
✓	2011-11-15 08:39 UTC	44	0	0	44	0	6	0	8.91 s	45.18 s
✓	2011-11-15 08:08 UTC	44	0	0	44	0	6	0	4.54 s	41.42 s
✓	2011-11-15 07:37 UTC	44	0	0	44	0	6	0	20.21 s	47.72 s
✓	2011-11-15 07:06 UTC	44	0	0	44	0	6	0	7.67 s	33.59 s
✓	2011-11-15 06:35 UTC	44	0	0	44	0	6	0	7.52 s	38.60 s
✓	2011-11-15 06:05 UTC	44	0	0	44	0	6	0	10.06 s	23.20 s
✓	2011-11-15 05:35 UTC	44	0	0	44	0	6	0	2.31 s	2.89 s

More »

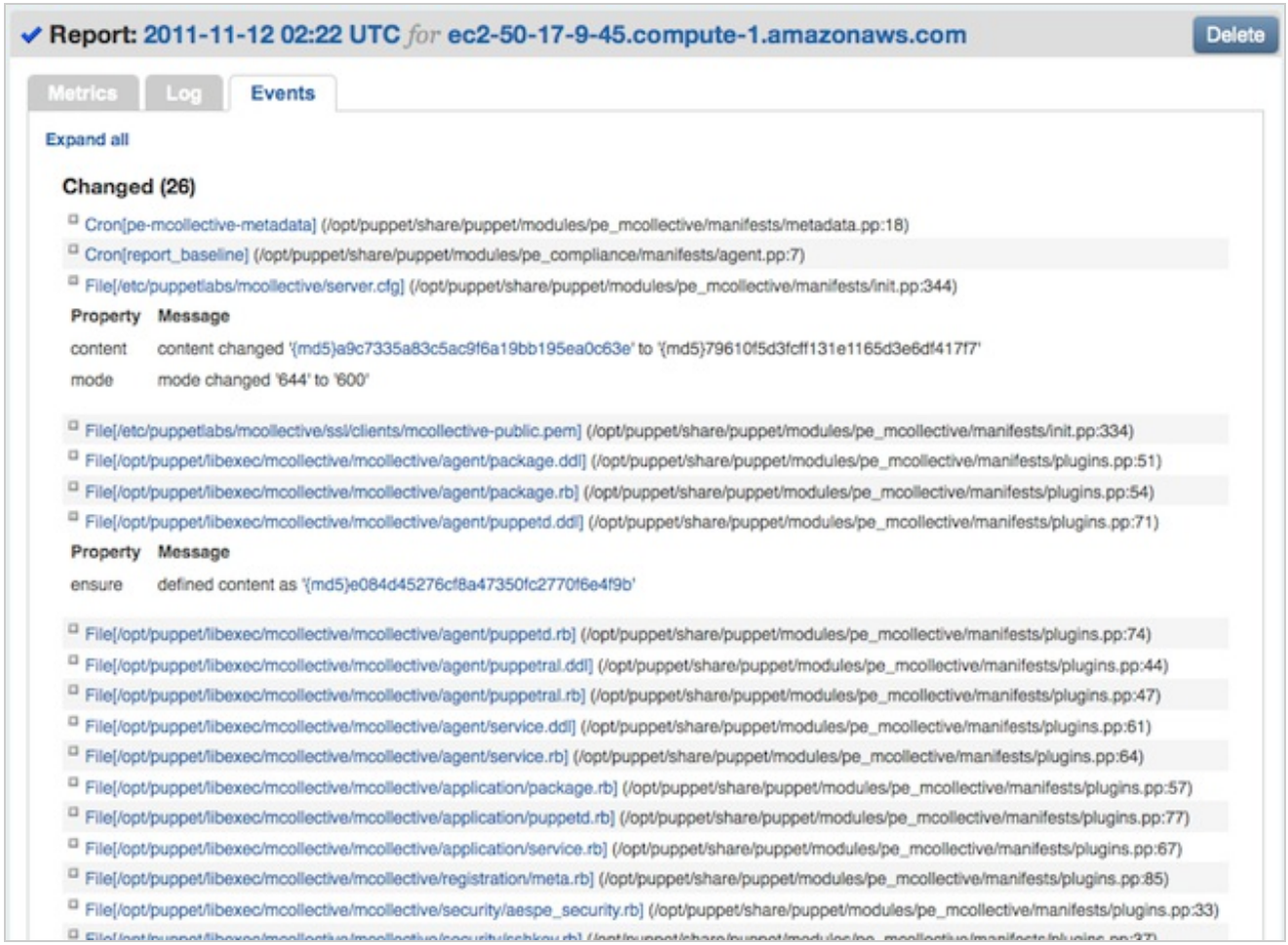
Each report represents a single Puppet run. Clicking a report will take you to a tabbed view that splits the report up into metrics, log, and events.

Metrics is a rough summary of what happened during the run, with resource totals and the time spent retrieving the configuration and acting on each resource type.□

Log is a table of all the messages logged during the run.

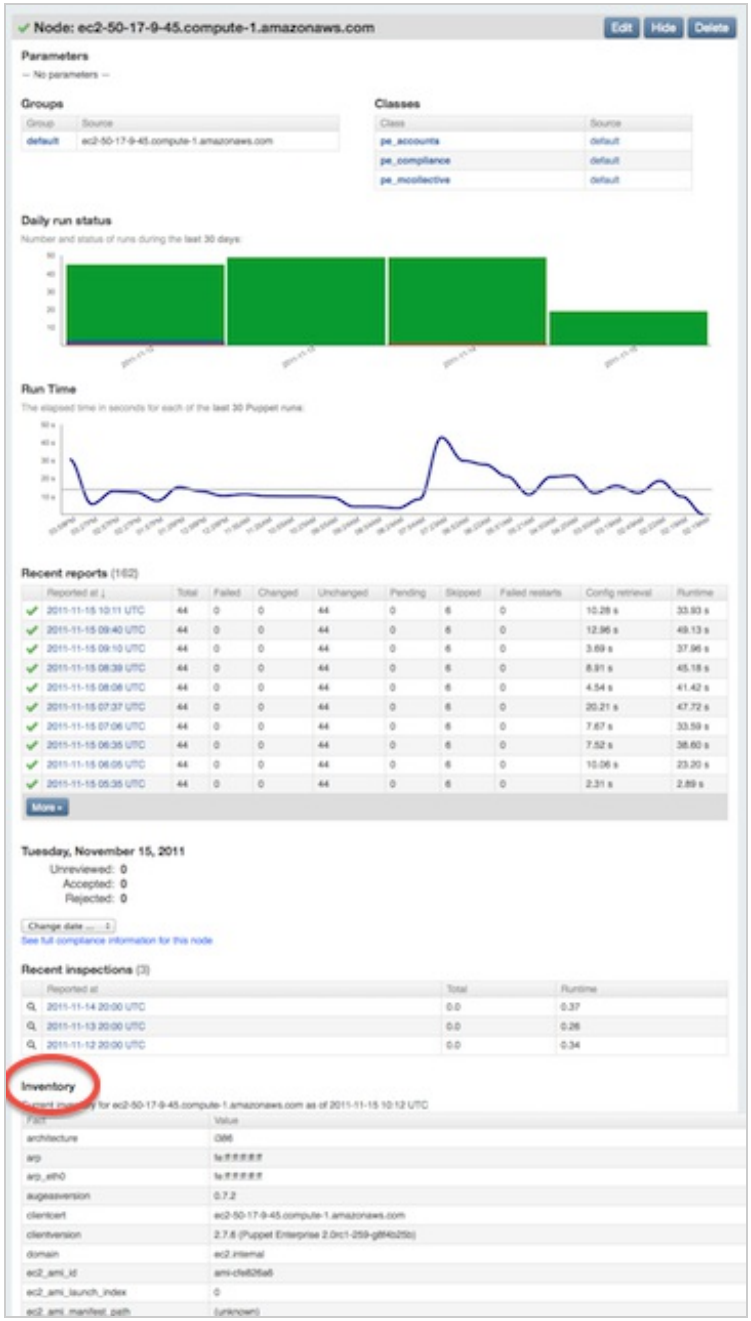


Events is a list of the resources the run managed, sorted by whether any changes were made. You can click on a changed resource to see which attributes were modified.



Viewing Inventory Data

Each node's page has a section called inventory. This section contains all of the fact values reported by the node on its most recent run.



Inventory

Current inventory for ec2-50-17-9-45.compute-1.amazonaws.com as of 2011-11-15 10:12 UTC

Fact	Value
architecture	i386
arp	fe:ff:ff:ff:ff:ff
arp_eth0	fe:ff:ff:ff:ff:ff
augeasversion	0.7.2
clientcert	ec2-50-17-9-45.compute-1.amazonaws.com
clientversion	2.7.6 (Puppet Enterprise 2.0rc1-259-g8f4b25b)
domain	ec2.internal
ec2_ami_id	ami-cfe826a6
ec2_ami_launch_index	0
ec2_ami_manifest_path	(unknown)

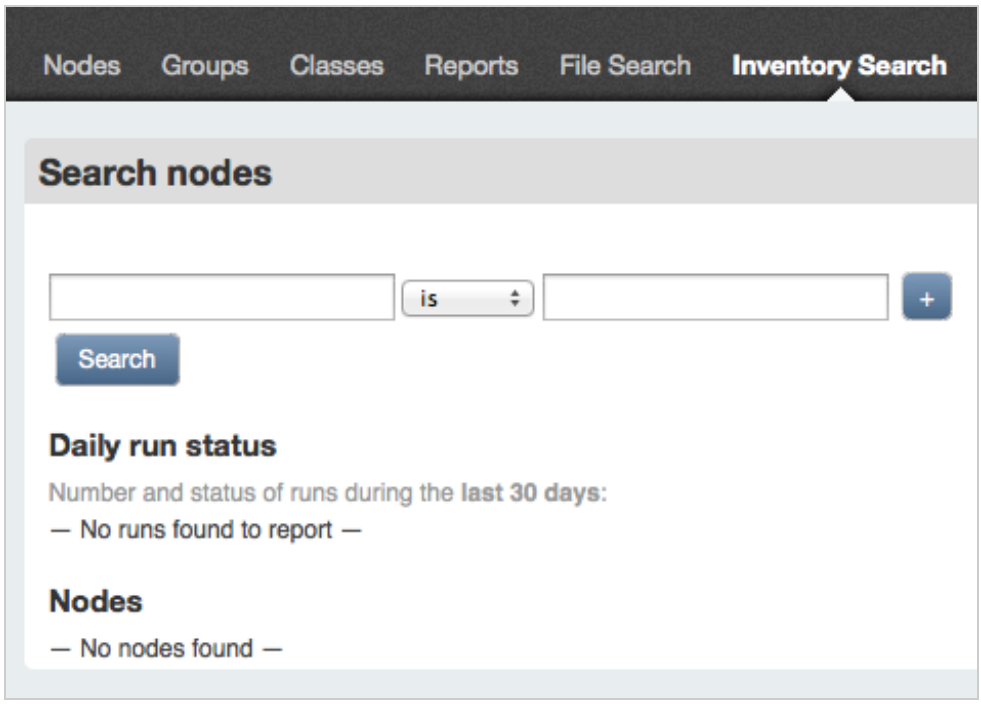
ec2_block_device_mapping_ami	/dev/sda1
ec2_block_device_mapping_ephemeral0	/dev/sda2
ec2_block_device_mapping_root	/dev/sda1
ec2_hostname	ip-10-202-214-228.ec2.internal
ec2_instance_id	i-353eb456
ec2_instance_type	t1.micro
ec2_kernel_id	aki-407d9529
ec2_local_hostname	ip-10-202-214-228.ec2.internal
ec2_local_ipv4	10.202.214.228
ec2_placement_availability_zone	us-east-1a
ec2_profile	default-paravirtual
ec2_public_hostname	ec2-50-17-9-45.compute-1.amazonaws.com
ec2_public_ipv4	50.17.9.45
ec2_public_keys_0_openssh_key	ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAnU9R6RuXDomfhwsVPq56Y8 /dFPRIyuvm+hU39a+dhMc+AuTqrw9kpVLKdA4pyqkeW5remwi+MskutDtGqsV
ec2_reservation_id	r-647c5d0a
ec2_security_groups	default
environment	production
fact_is_puppetagent	true
fact_is_puppetconsole	false
fact_is_puppetmaster	false
fact_stomp_port	61613
fact_stomp_server	ec2-50-16-137-167.compute-1.amazonaws.com
facterversion	1.6.2
fqdn	ip-10-202-214-228.ec2.internal
hardwareisa	unknown
hardwaremodel	i686
hostname	ip-10-202-214-228
id	root
interfaces	dummy0,eql,eth0,ifb0,ifb1,lo

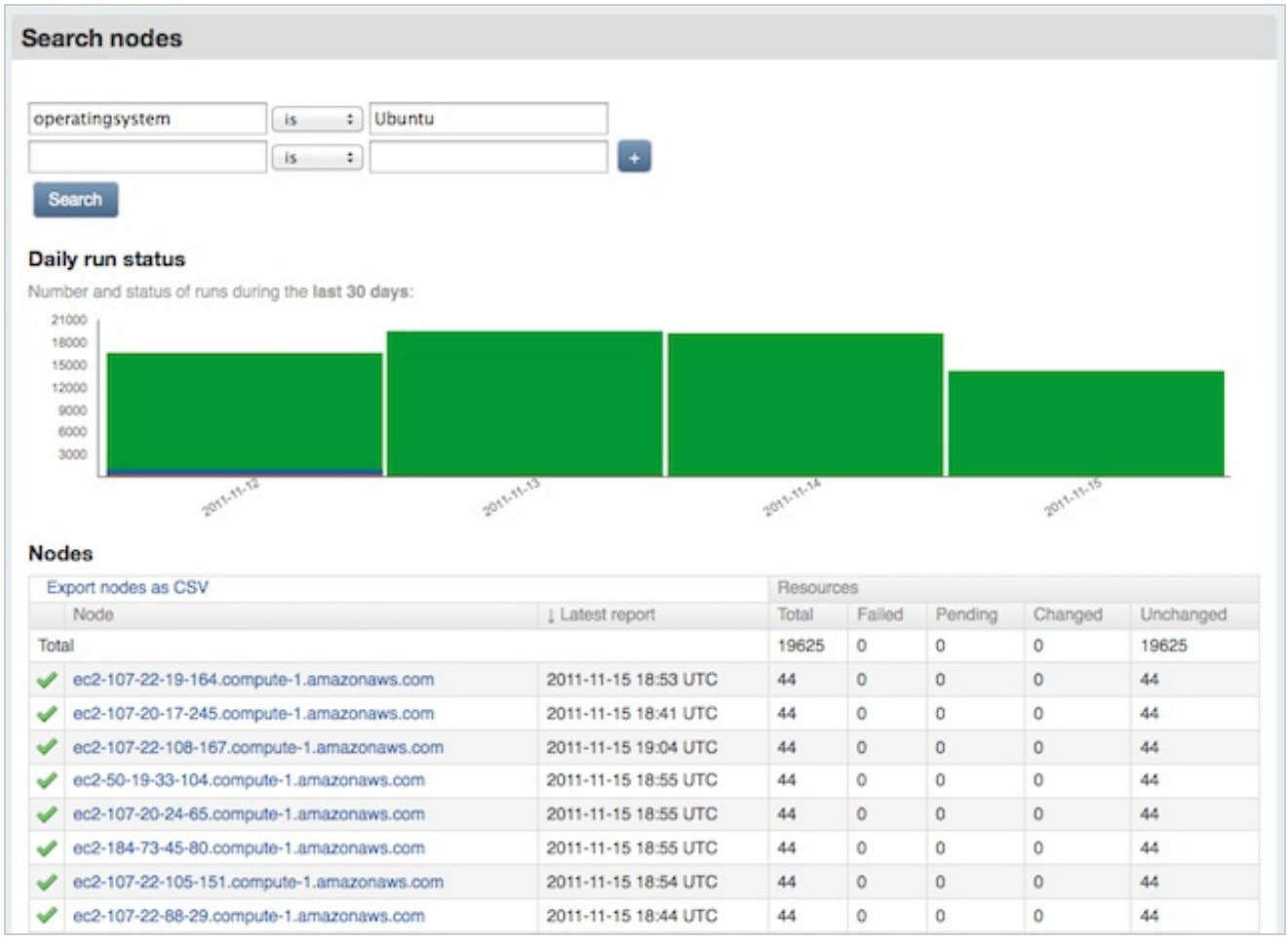
Facts include things like the operating system (`operatingsystem`), the amount of memory (`memorytotal`), and the primary IP address (`ipaddress`). You can also add arbitrary custom facts to your Puppet modules, and they too will show up in the inventory.

The facts you see in the inventory can be useful when filtering nodes in the live management page.

Searching by Fact

Use the “inventory search” page to find a list of nodes with a certain fact value.





You can add more facts to filter the search results further, and can change the comparison criteria for each one.

← Console: Navigating the Console — Index — Console: Grouping and Classifying Nodes →

PE 2.0 » Console » Grouping and Classifying Nodes

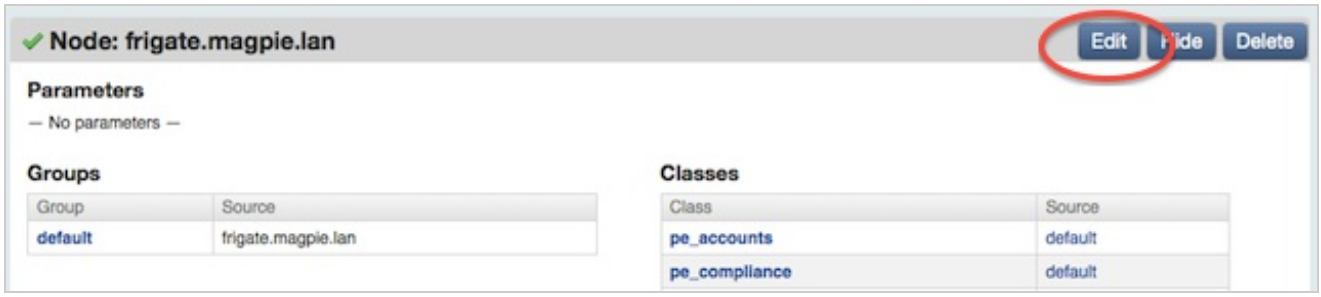
Grouping and Classifying Nodes

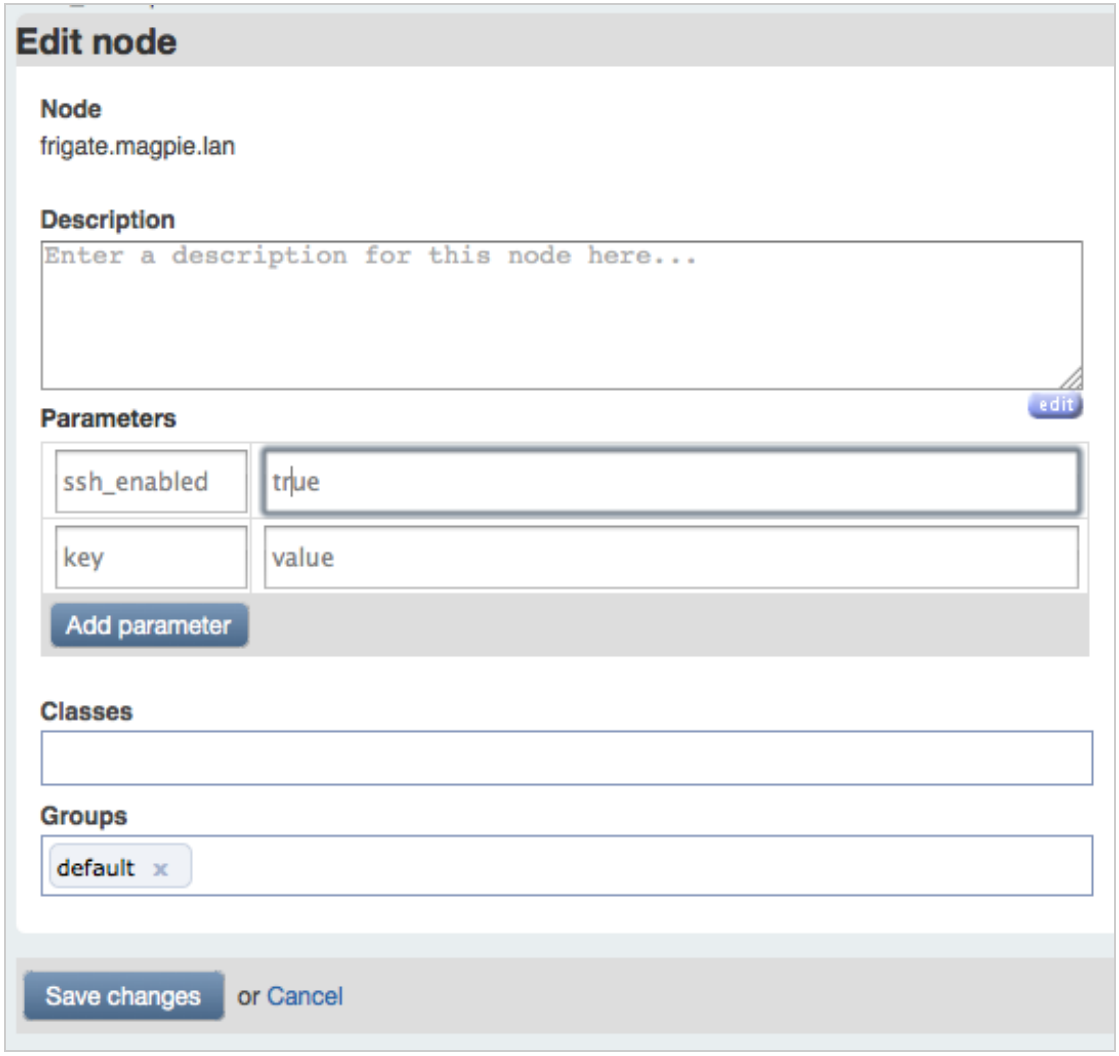
Use groups, classes, and parameters to control which Puppet configurations your nodes receive.

Parameters

Parameters are simple: they're top-scope variables that your Puppet manifests can use. Use them to configure the behavior of classes.

Add parameters by clicking the edit button in a node view (or group view) and typing a key and value under the "parameters" heading. Use the "add parameter" button below to make additional key/value fields. Be sure to save your changes when done.





Parameters can only be strings, not arrays or hashes.

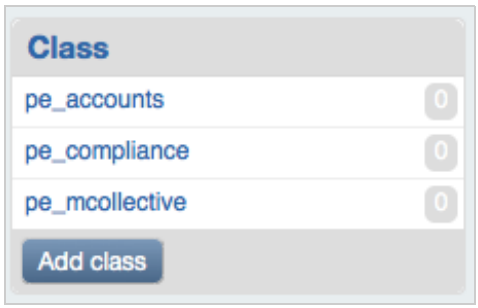
Classes

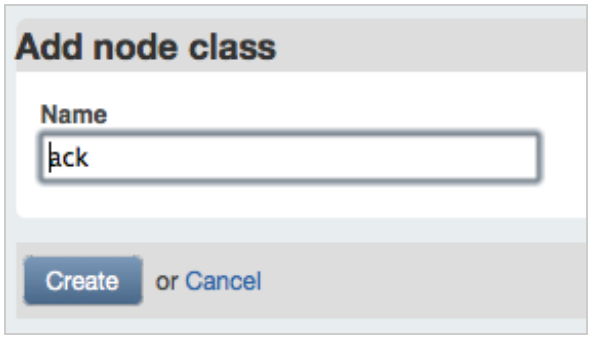
The classes the console knows about are a subset of the classes in your puppet master's collection of modules. You must add classes to the console manually if you want to assign them to any nodes or groups.

See the Puppet section of this user's guide for an introduction to Puppet classes.

Adding a New Class

Use the "Add class" button in the console's sidebar, then type the new class's name in the text field and click "create."



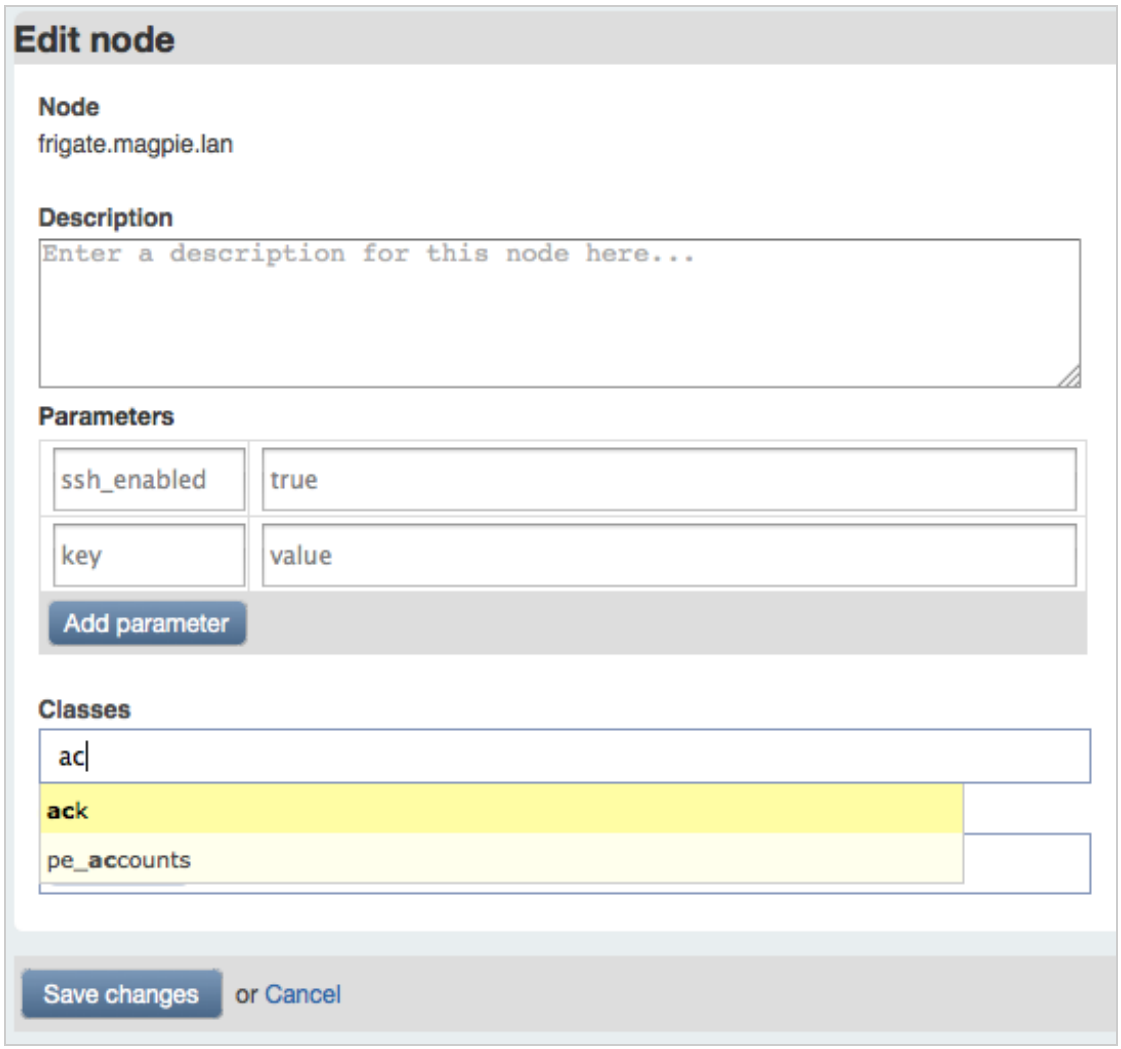


In this case, we're adding a tiny class that makes sure ack is present in `/usr/local/bin`.

When adding a class, you must use its fully qualified name. (`base::centos`, for example.)

Assigning a Class to a Node

Assign classes by clicking the edit button in a node view (or group view). Start typing the class's name in the "classes" field, then choose the class you want from the auto-completion list. Be sure to save your changes when done.



Writing Classes for the Console

Defining wrapper classes in a "site" module can help you use the console more effectively, and may be mandatory if you use parameterized classes heavily.

Most Puppet modules are written so each class manages a logical chunk of configuration. This means any node's configuration could be composed of dozens of Puppet classes. Although you can add these dozens of classes to the console, it's often better to create a module called `site` and populate it with super-classes, which declare all of the smaller classes a given type of machine will need.

There are many ways to compose these classes, and you'll have to decide based on how your own collection of modules works. Some possibilities:

- Create many non-overlapping classes, such that any node will only have one class assigned to it.
- Create several separated "levels" of classes — a "base" layer, a layer for role-specific packages and services, a layer for application data, etc. This way, each node can get the base class for its own OS or machine type, but use the same application classes as some other quite different node.

Wrapper classes are also necessary for working with parameterized classes — you can declare

parameters in nodes and groups, then have your wrapper classes pass them through when they declare each smaller class.

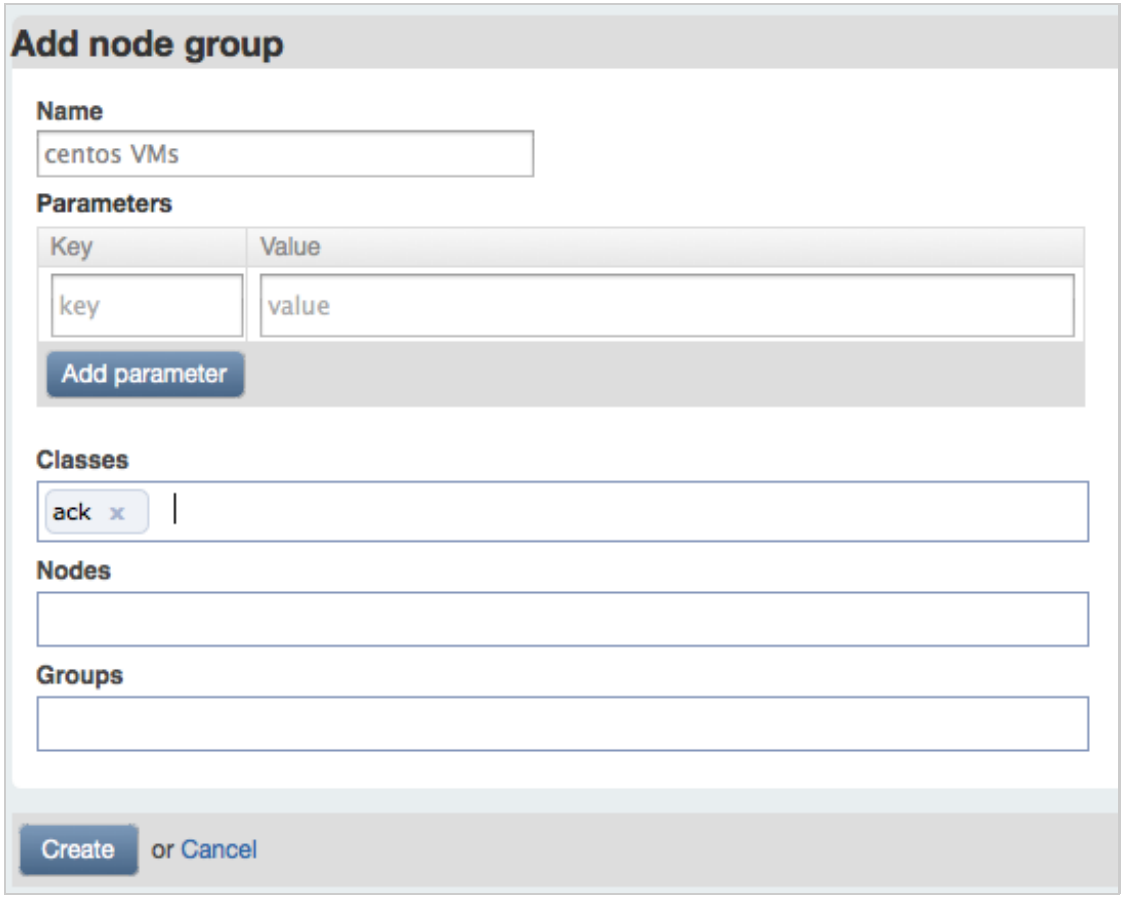
Grouping Nodes

Groups let you assign a class or parameter to many nodes at once. This saves you time and makes the structure of your site more knowable.

Nodes can belong to many groups, and inherit classes and parameters from all of them. Groups can also contain other groups, which will inherit information the same way nodes do.

Adding a New Group

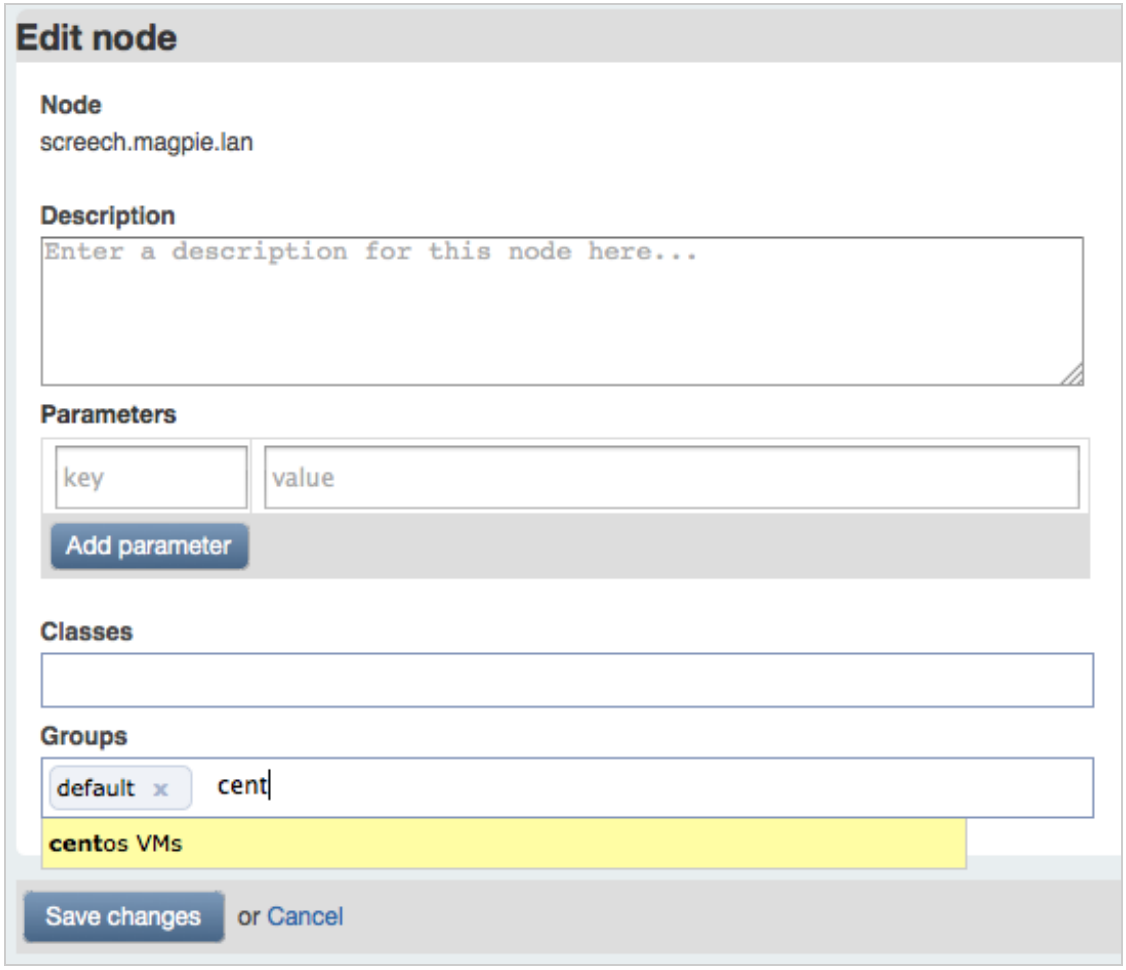
Use the “add group” button in the console’s sidebar, then enter the group’s name and any classes or parameters you want to assign.

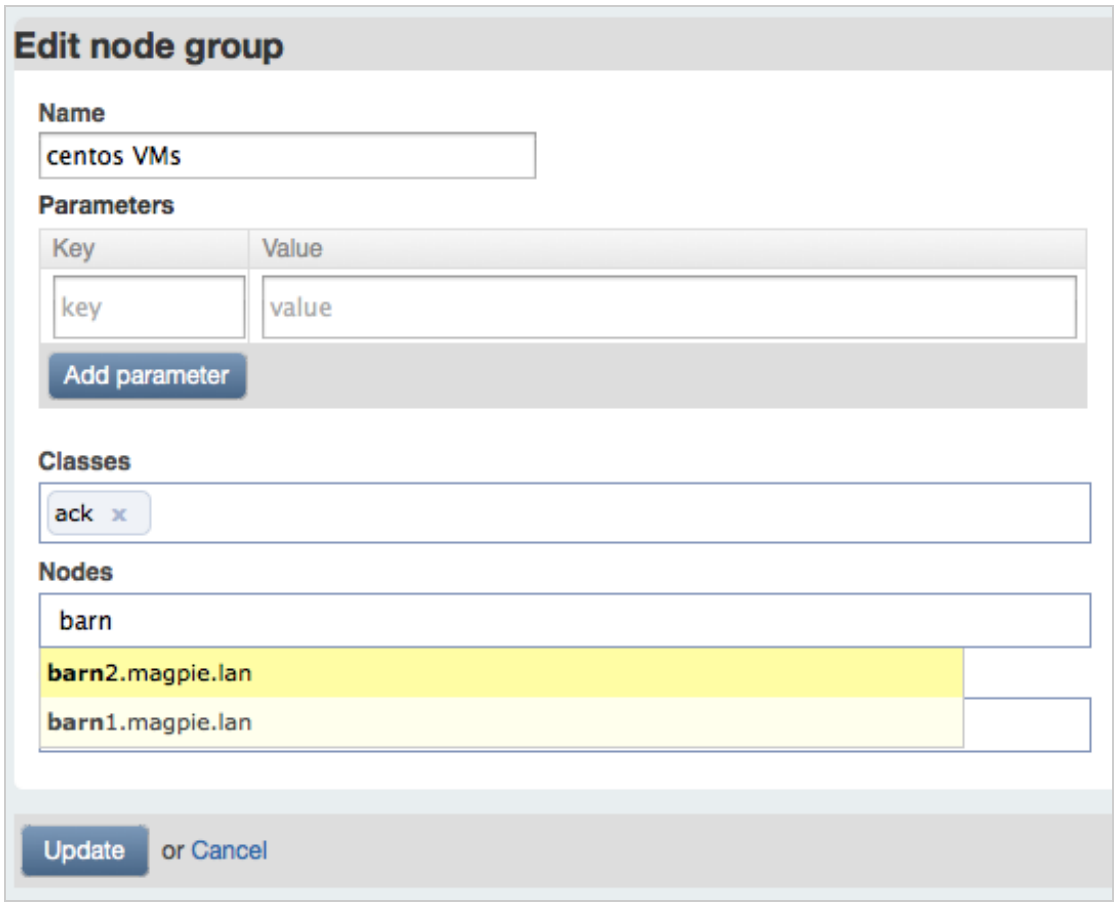


Adding Nodes to a Group

You can change the membership of a group from both node views and group views. Click the edit button and use the “groups” or “nodes” fields, as needed. These fields will offer auto-completions□

the same way the classes field does.□





Assigning Classes and Parameters to a Group

This works identically to assigning classes and parameters to a single node. Use the edit button and the classes or key/value fields.

The Default Group

The console automatically adds every node to a group called `default`. Use this group for any classes you need assigned to every single node. This differs from the open source Dashboard, which doesn't currently implement a default group.

Nodes are added to the default group by a periodic background task, so it may take several minutes after a node first checks in before it joins the group.

Rake API

The console provides rake tasks that can group nodes, create classes, and assign classes to groups. You can use these as an API to automate workflows or bypass the console's GUI when performing large tasks.

All of these tasks should be run as follows, replacing `<TASK>` with the task name and any arguments it requires:

```
# sudo /opt/puppet/bin/rake -f /opt/puppet/share/puppet-dashboard/Rakefile
<TASK>
```

Node Tasks

`node:list [match=<REGULAR EXPRESSION>]`
List nodes. Can optionally match nodes by regex.
`node:add name=<NAME> [groups=<GROUPS>] [classes=<CLASSES>]`
Add a new node. Classes and groups can be specified as comma-separated lists.
`node:del name=<NAME>`
Delete a node.
`node:classes name=<NAME> classes=<CLASSES>`
Replace the list of classes assigned to a node. Classes must be specified as a comma-separated list.
`node:groups name=<NAME> groups=<GROUPS>`
Replace the list of groups a node belongs to. Groups must be specified as a comma-separated list.

Class Tasks

`nodeclass:list [match=<REGULAR EXPRESSION>]`
List node classes. Can optionally match classes by regex.
`nodeclass:add name=<NAME>`
Add a new class. This must be a class available to the Puppet autoloader via a module.
`nodeclass:del name=<NAME>`
Delete a node class.

Group Tasks

`nodegroup:list [match=<REGULAR EXPRESSION>]`
List node groups. Can optionally match groups by regex.
`nodegroup:add name=<NAME> [classes=<CLASSES>]`
Create a new node group. Classes can be specified as a comma-separated list.
`nodegroup:del name=<NAME>`
Delete a node group.
`nodegroup:add_all_nodes name=<NAME>`
Add every known node to a group.
`nodegroup:addclass name=<NAME> class=<CLASS>`
Assign a class to a group without overwriting its existing classes.
`nodegroup:edit name=<NAME> classes=<CLASSES>`
Replace the classes assigned to a node group. Classes must be specified as a comma-separated list.

← Console: Viewing Reports and Inventory Data — Index — Console: Live Management →

PE 2.0 » Console » Live Management

← Console: Grouping and Classifying Nodes — Index — Console: Live Management: Managing Resources →

Live Management

What's Live Management?

The console's live management page contains tools for inspecting and editing your nodes in real time. It is powered by MCollective.

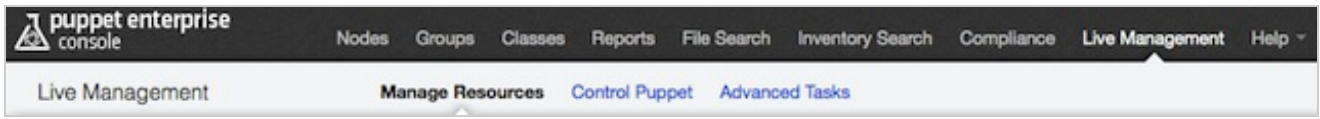
Since the live management page queries information directly from your nodes rather than using the console's cached reports, it responds more slowly than other parts of the console.

Show Me!

Puppet Labs' own Randall Hansen demonstrates the power of live management:

Tabs

The live management page is split into three tabs, one for each of its tools.

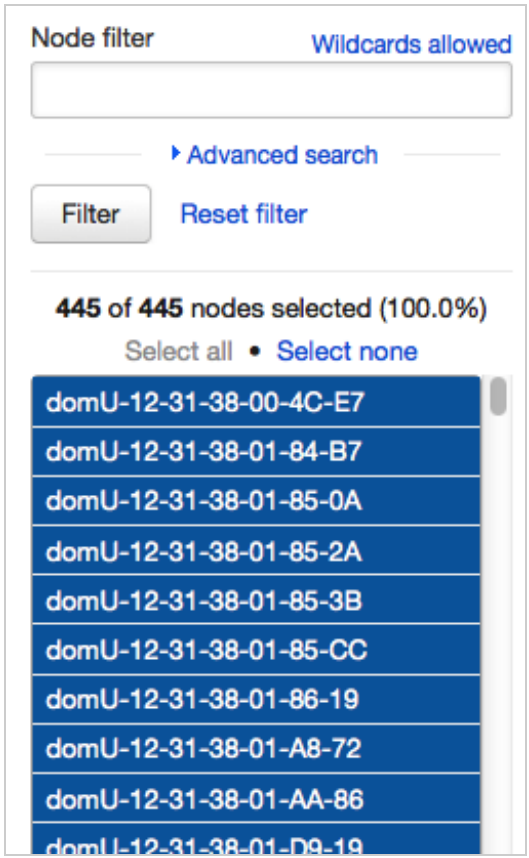


- The manage resources tab lets you browse the resources on your nodes and clone any of them across your infrastructure.
- The control Puppet tab lets you tell any node to immediately pull and apply its configuration. It can also temporarily disable puppet agent on some of your nodes to control the rollout speed of new configurations.
- The advanced tasks tab is a direct interface to the MCollective agents on your systems, and will auto-generate a GUI for any new agents you install.

The following chapters of this section cover the use of each tab.

The Node List

Every task in live management inspects or modifies a selection of nodes. Use the node list in the live management sidebar to choose the nodes for your next action. (This list will only contain nodes that have completed at least one Puppet run, which may take up to 30 minutes after you've signed its certificate.)



Nodes are listed by their hostnames in the live management pages. A node's hostname may match the name Puppet knows it by, but this isn't necessarily the case, especially under cloud environments like EC2.

As long as you stay within the live management page, your selection and filtering in the node list will persist across all three tabs. The node list gets reset once you navigate to a different area of the console.

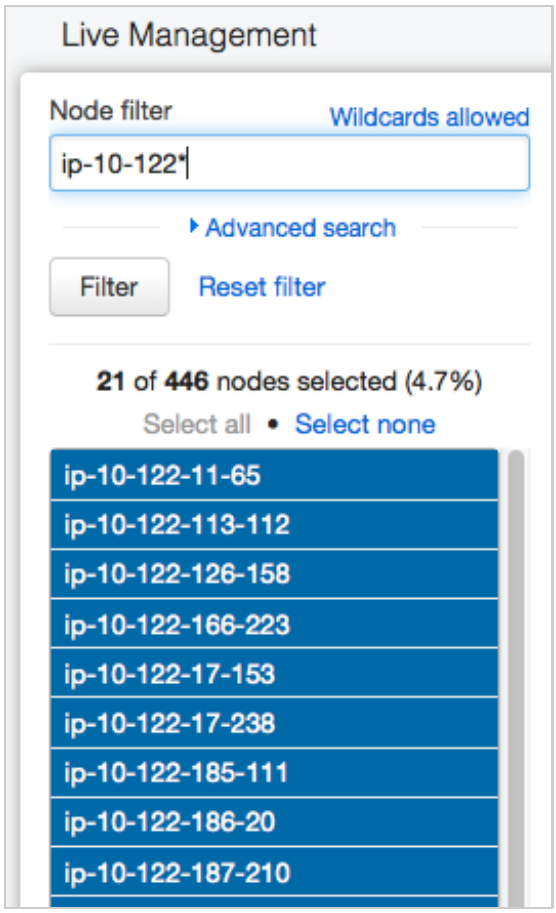
Selecting Nodes

Clicking a node selects it or deselects it. Use the "select all" and "select none" controls to select and deselect all nodes that match the current filter.

Only nodes that match the current filter can be selected; you don't have to worry about acidentally modifying "invisibly" selected nodes.

Filtering by Name

Use the “node filter” field to filter your nodes by hostname. This type of filtering is most useful for small numbers of nodes, or for situations where your hostnames have been assigned according to some logical plan.



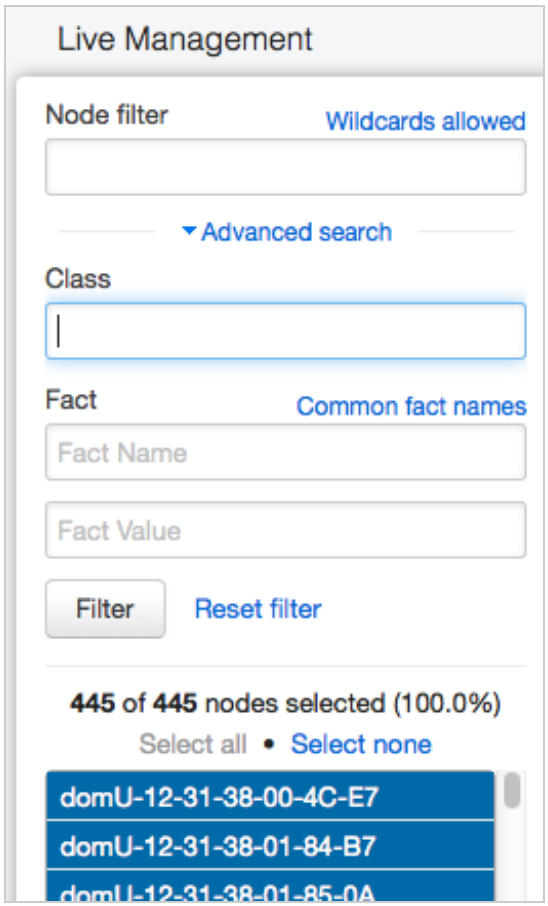
You can use the following wildcards in the node filter field:

- ? matches one character
- * matches many (or zero) characters

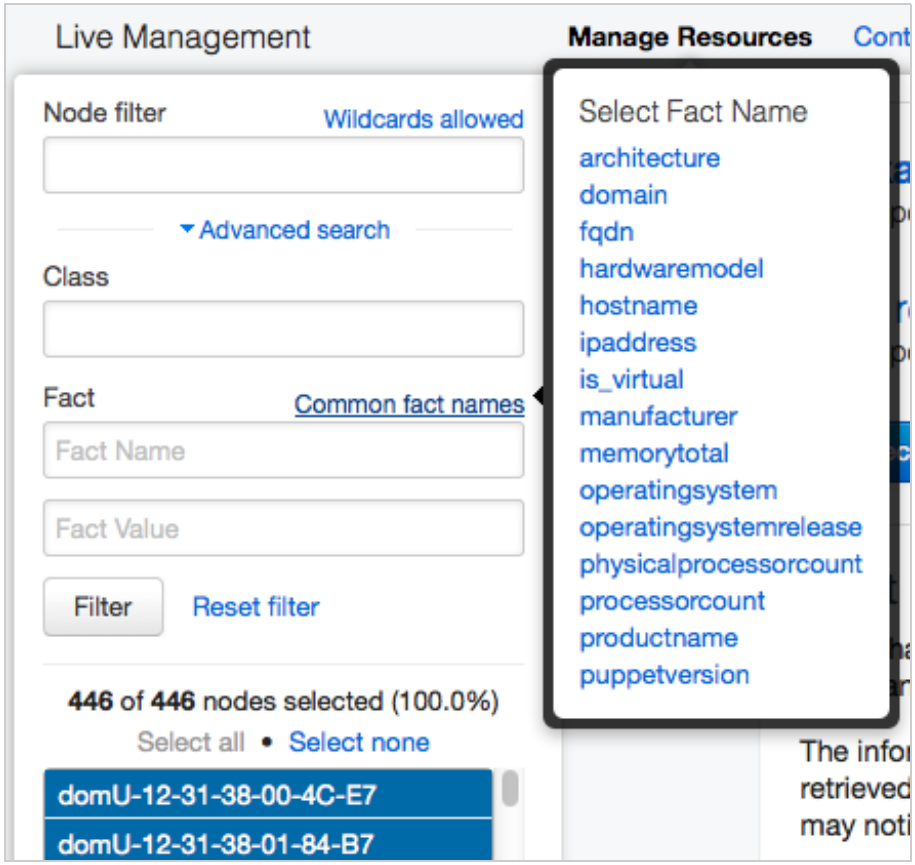
Use the “filter” button or the enter key to confirm your search, then wait for the node list to be updated.

Advanced Search

You can also filter by Puppet class or by the value of any fact on your nodes. Click the “advanced search” link to reveal these fields.



You can select from some of the more useful fact names with the “common fact names” popover:



The inventory data in the console’s node views is another good source of facts to search with.

Filtering by Puppet class can be the most powerful filtering tool on this page, but it requires you to have already sorted your nodes by assigning distinctive classes to them. See the chapter on grouping and classifying nodes for more details.

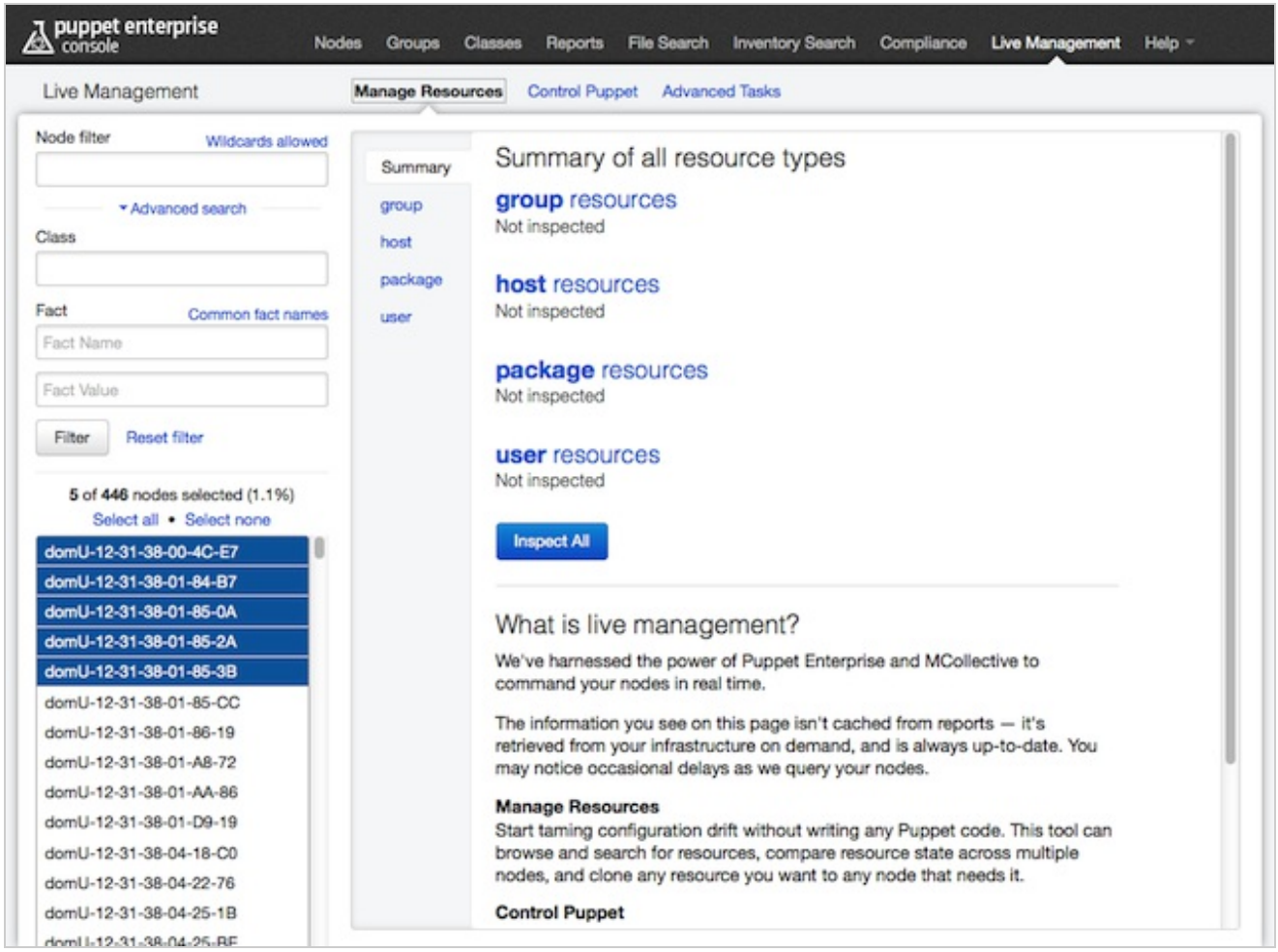
← Console: Grouping and Classifying Nodes — Index — Console: Live Management: Managing Resources →

PE 2.0 » Console » Live Mgmt: Managing Resources

← Console: Live Management — Index — Console: Live Management: Controlling Puppet →

Live Management: Managing Resources

Use the manage resources tab to browse the resources on your nodes and clone any of them across your infrastructure.



Resource Types

The live management tools can manage the following resource types:

- user
- group
- host
- package

For an introduction to resources and types, please see the Resources chapter of Learning Puppet.

The Summary View

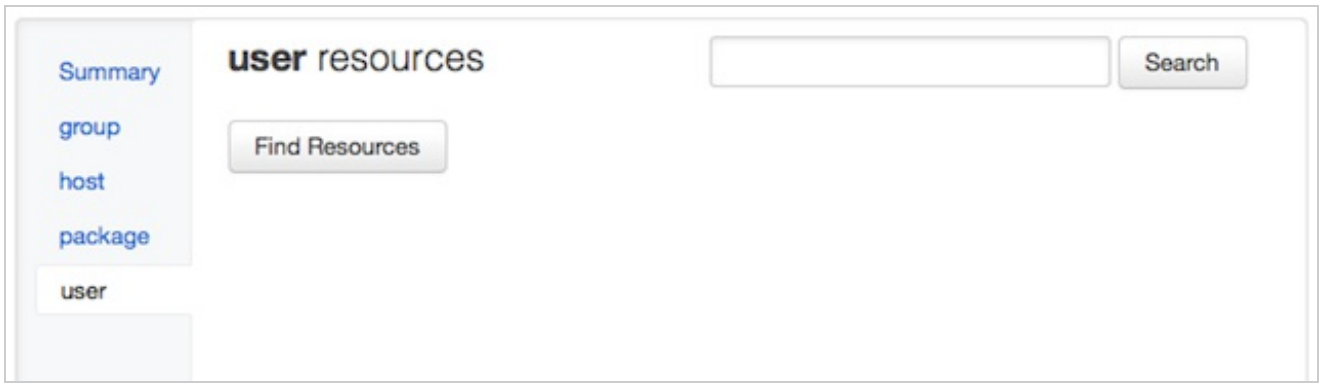
The summary view has an “Inspect All” button, which scans all resources of all types and reports on their similarity. This is mostly useful when you think you’ve selected a group of identical nodes but want to make sure.

Using “Inspect All” at the start of a session will pre-load a lot of information into memory, which can speed up later operations.

Finding Resources

To find a resource to work with, you must first select a resource type. Then, either search for your resource by name or load all resources of the type and browse them.

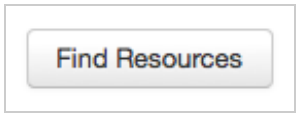
When you select a type for the first time in a session, it won’t automatically display any resources:

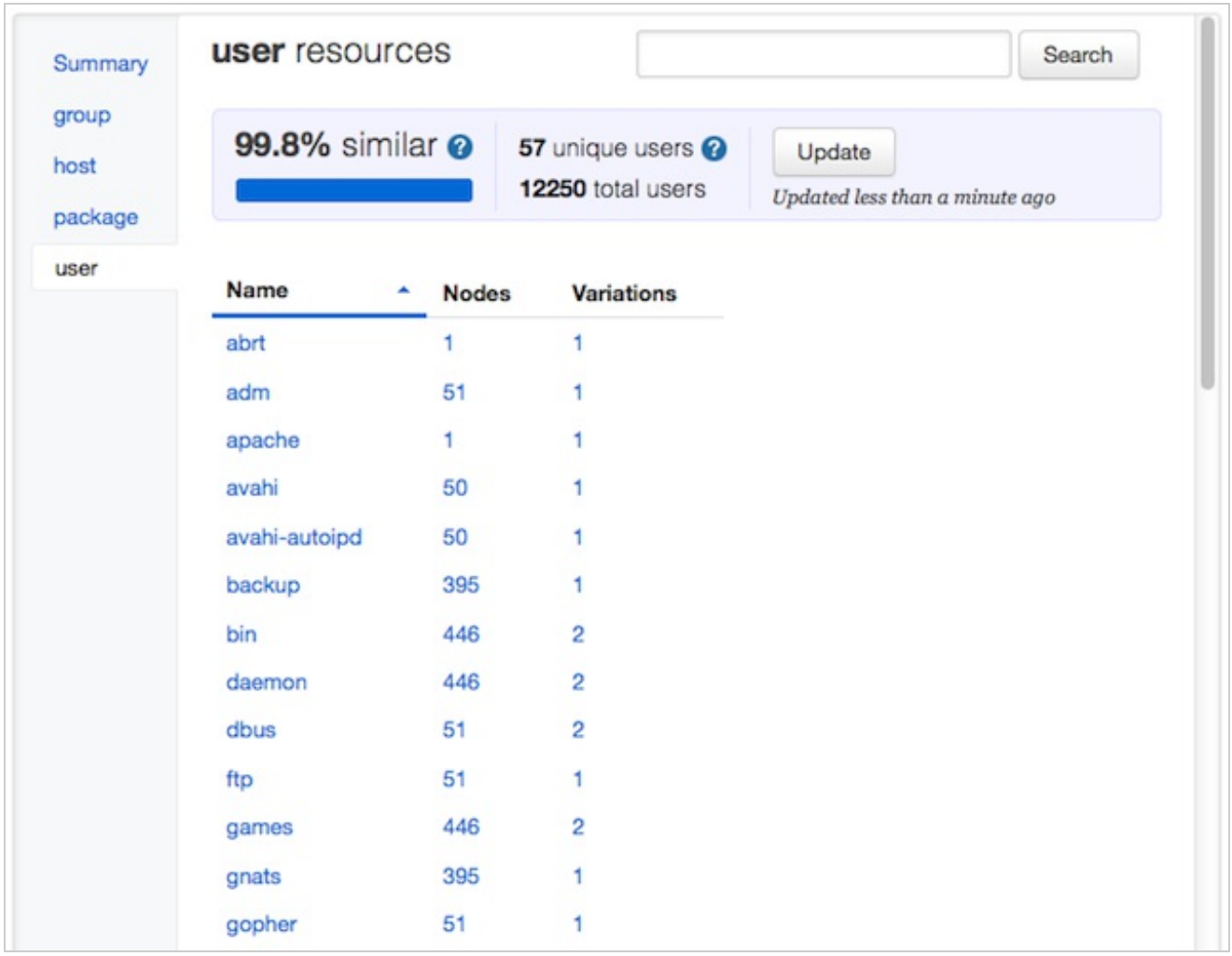


Searching and browsing only use the current selection of nodes.

Browsing a Type

To browse a list of all resources, use the “find resources” button.

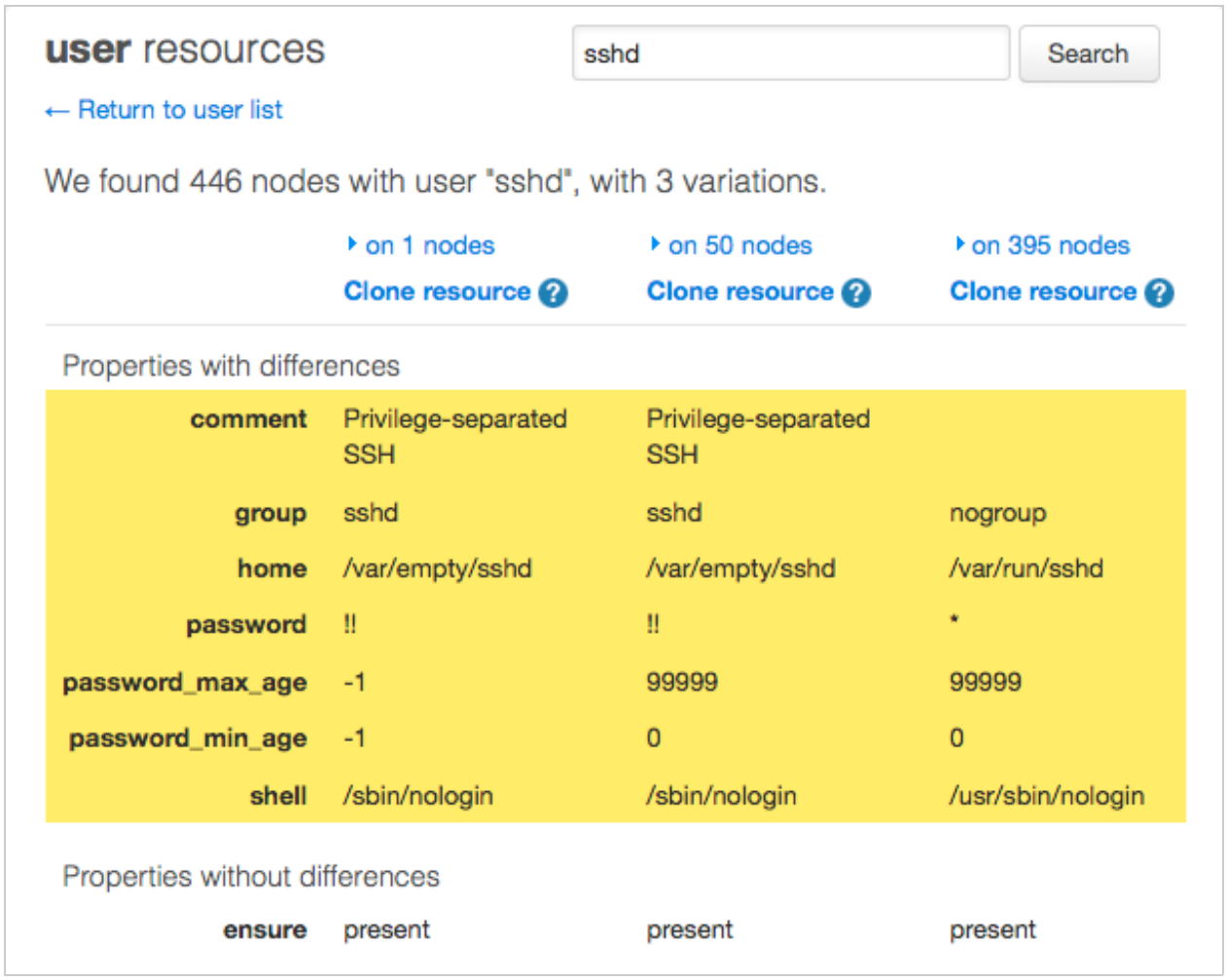




This will return a list of all resources of the selected type on the selected nodes, plus a summary of how similar the resources are. In general, a set of nodes that perform similar tasks should have very similar resources. You can make a set of nodes more similar by cloning resources across them.

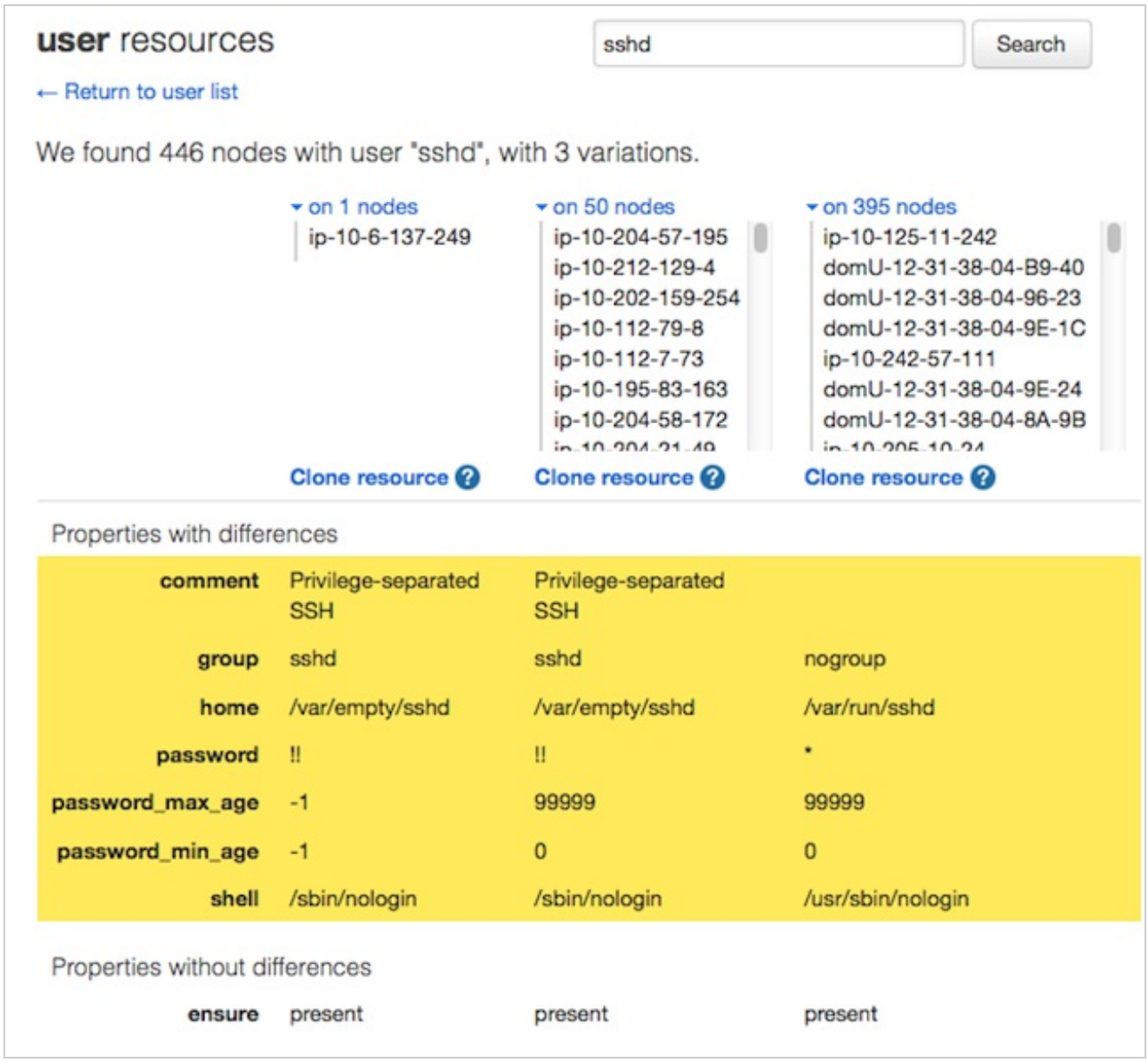
The resource list shows the name of each resource, the number of nodes it was found on, and how many variants of it were found. You can sort the list by any of these properties.

To inspect a resource, click its name.



When you inspect a resource, you can see the values of all its properties. If there is more than one variant, you can see all of them and the properties that differ will be highlighted.

To see which nodes have each variant, click the “on N nodes” labels to expand the node lists.

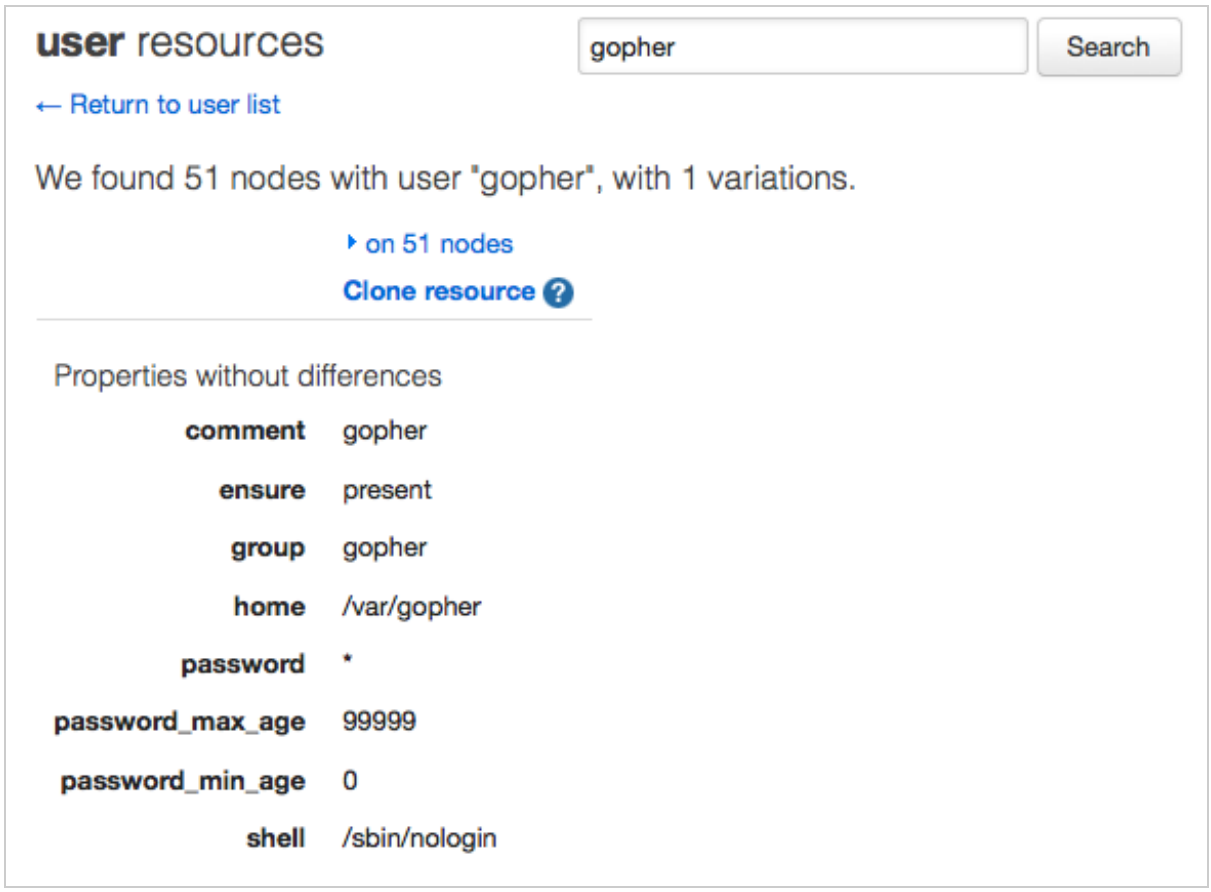


Searching by Name

To search, enter a resource name in the search field, and confirm with the enter key or the “search” button.

The name you search for has to be exact; wildcards are not allowed.

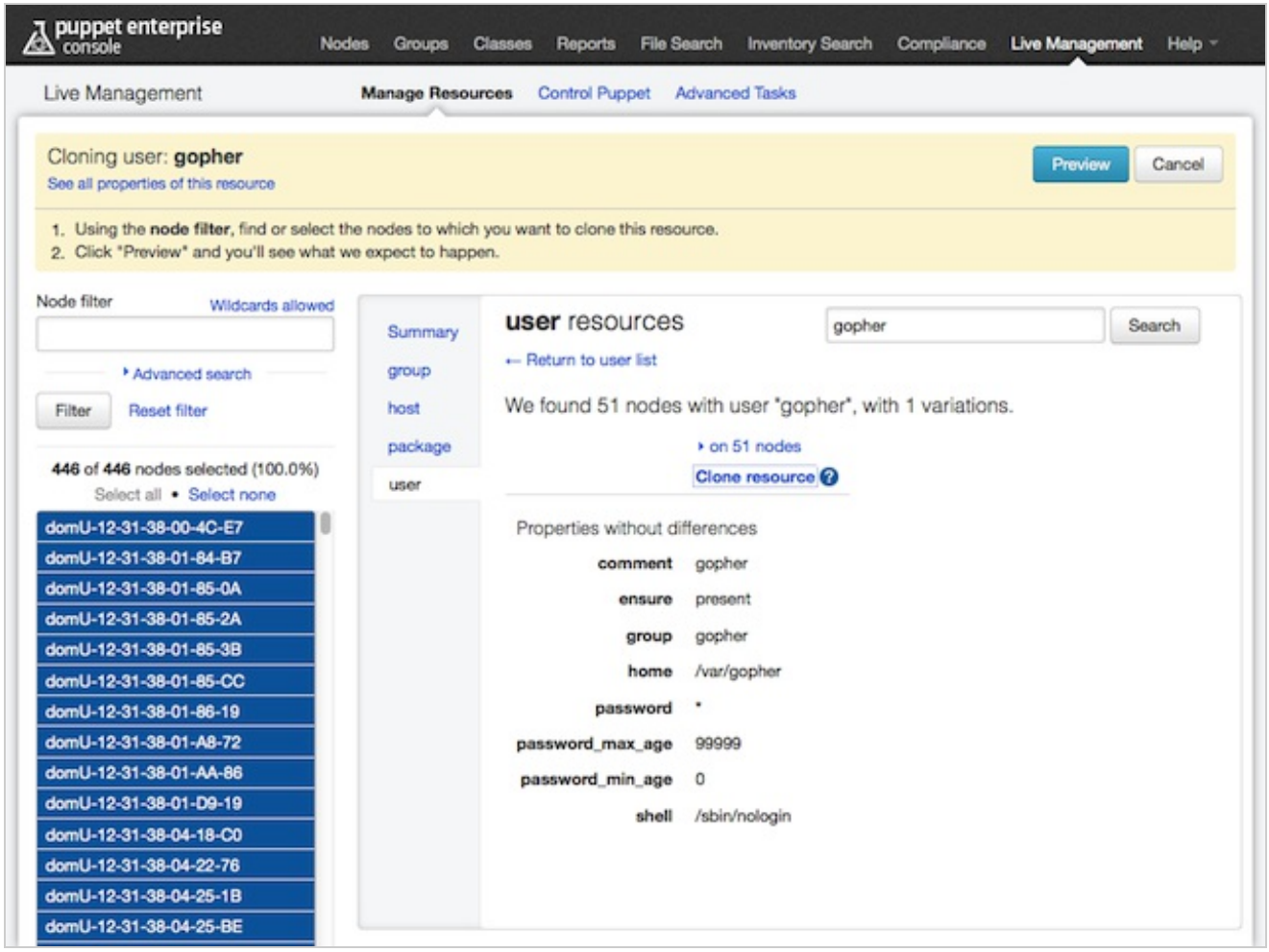
Once you’ve located a resource, the inspect view is the same as that used when browsing.



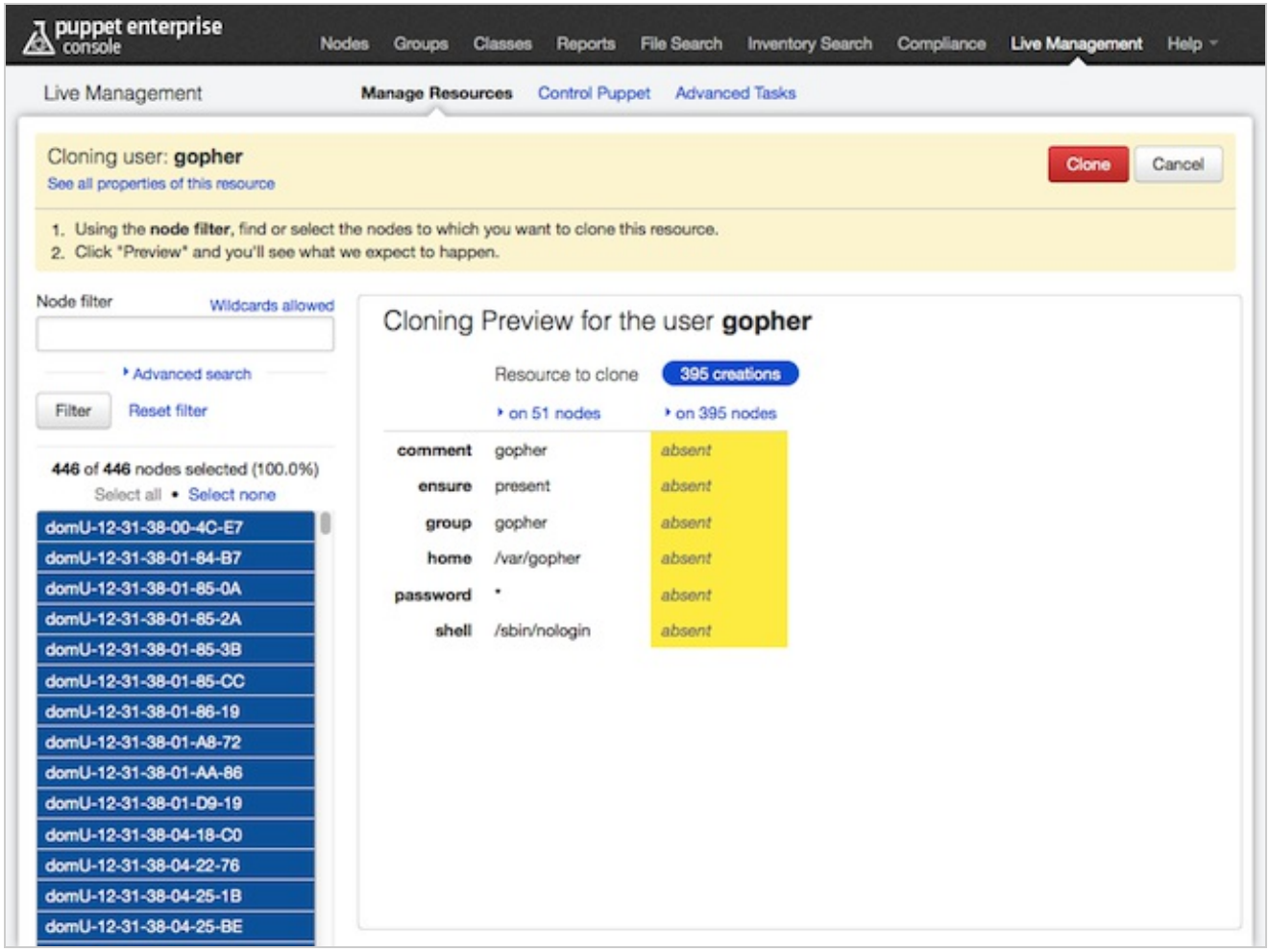
Cloning Resources

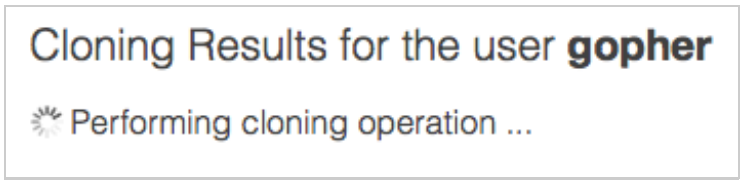
You can use the “clone resource” links on a resource’s inspect view to make it identical on all of the selected nodes. This lets you make your population of nodes more alike without having to write any Puppet code.

Clicking the clone link for one of the variants will raise a confirmation dialog. You can change the set of selected nodes before clicking the “preview” button.

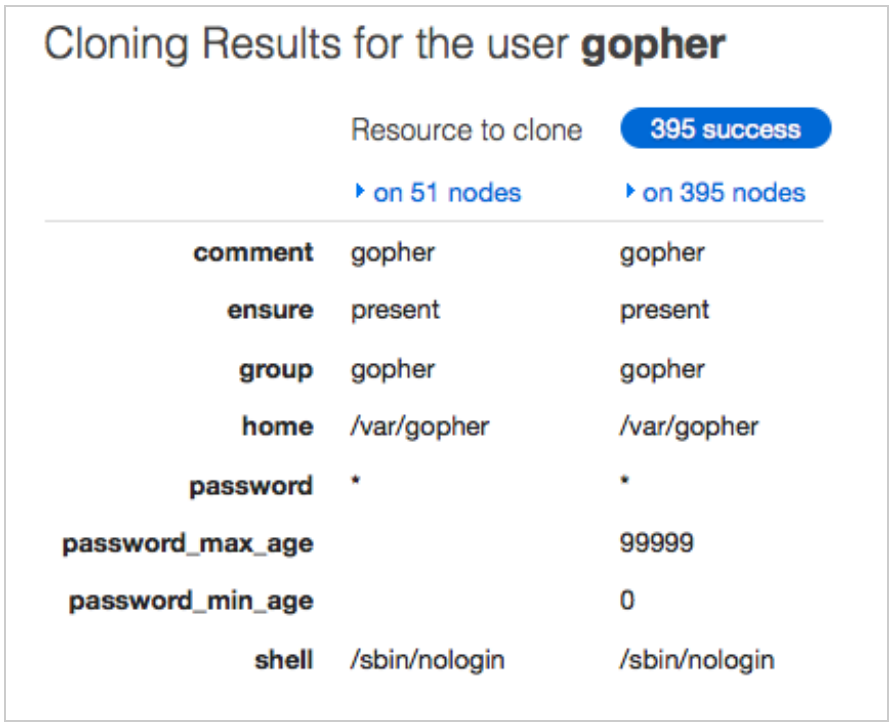


Clicking “preview” will show the pending changes and enable the “clone” button. If the changes look good, click “clone.”





After the resource has been cloned, you can see a summary of the results and the new state of the resource.



Special Behavior When Cloning Users

When you clone a user, any groups it belongs to are also automatically cloned.

Note also that the UID of a user and the GIDs of its groups aren't cloned across nodes; this means a cloned user's UID will likely differ across nodes. We hope to support UID/GID cloning in a future release.

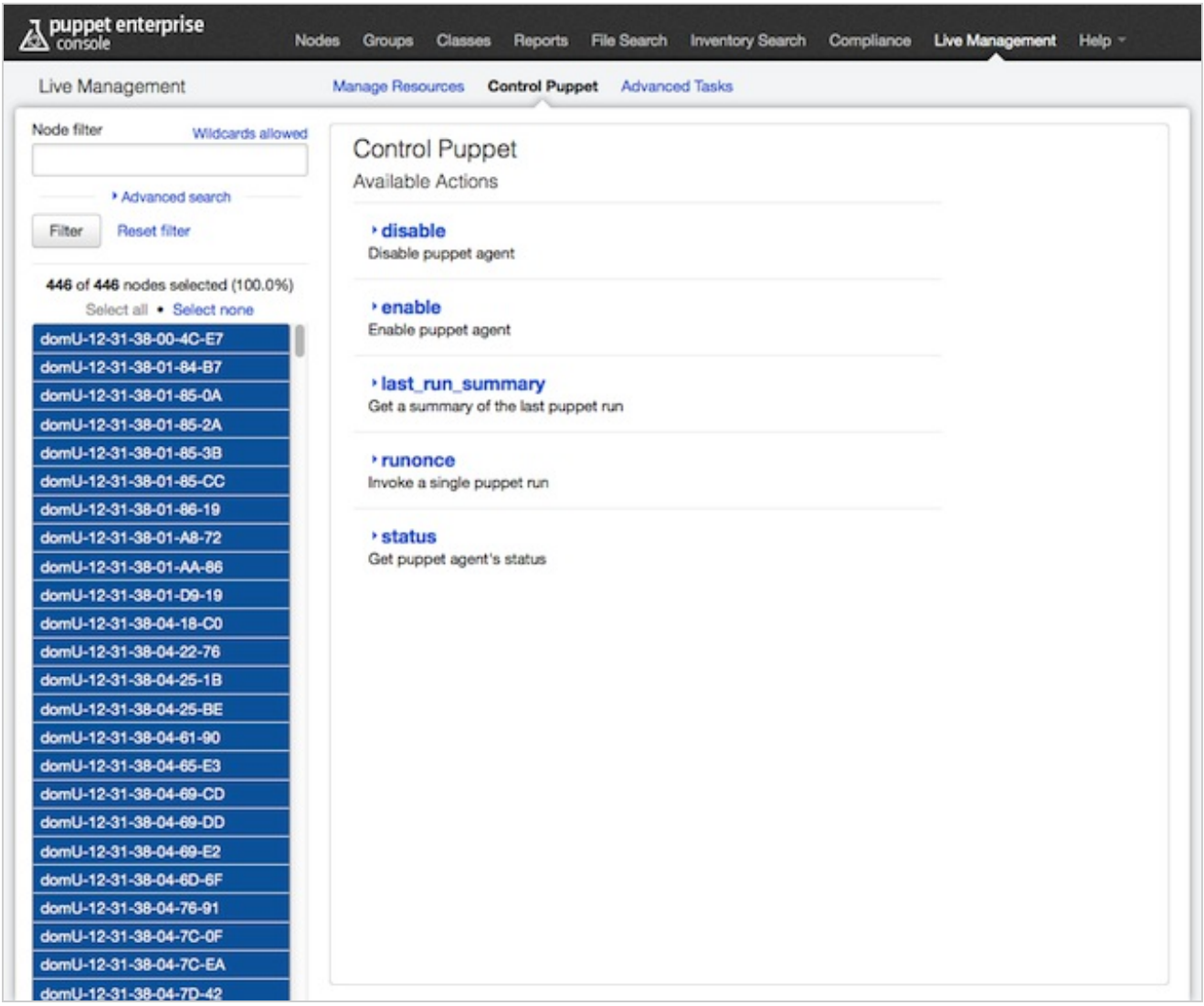
← Console: Live Management — Index — Console: Live Management: Controlling Puppet →

PE 2.0 » Console » Live Mgmt: Controlling Puppet

← Console: Live Management: Managing Resources — Index — Console: Live Management: Advanced Tasks →

Live Management: Controlling Puppet

Use the control puppet tab to immediately trigger a puppet agent run on any of your nodes. You can also check puppet agent's status, and enable or disable it to control the spread of new configurations.



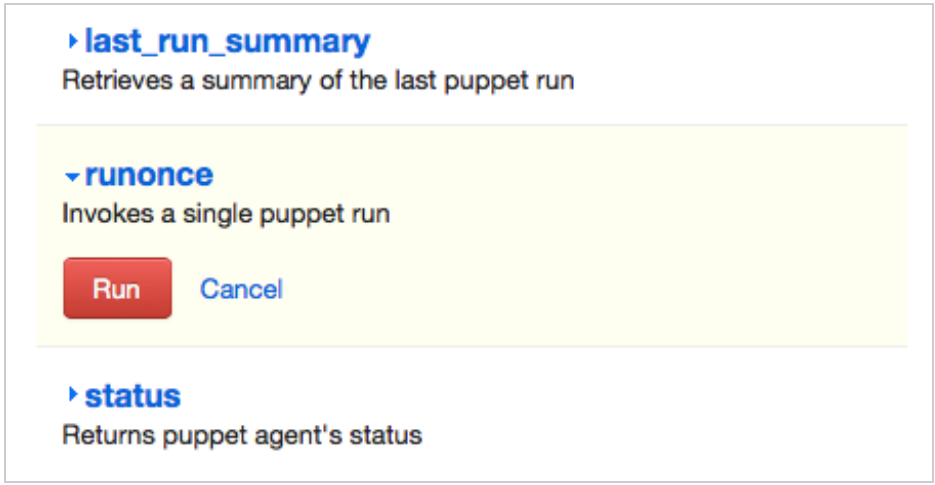
Note that the control puppet tab cannot trigger a node's first puppet agent run; a node's first run will happen automatically within 30 minutes after you sign its certificate.

Invoking an Action

The control puppet tab can perform five actions:

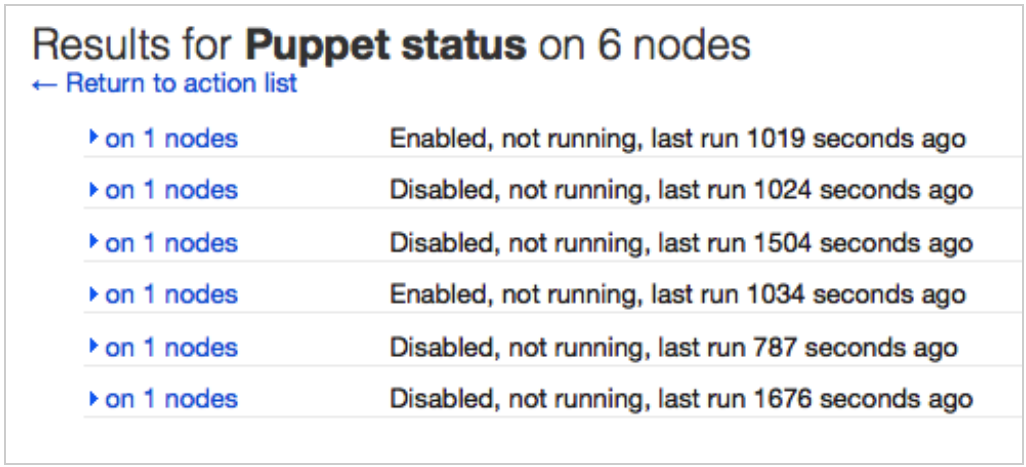
- Disable
- Enable
- Last Run Summary
- Runonce
- Status

To use one, click the name of the action, then confirm with the red “Run” button.

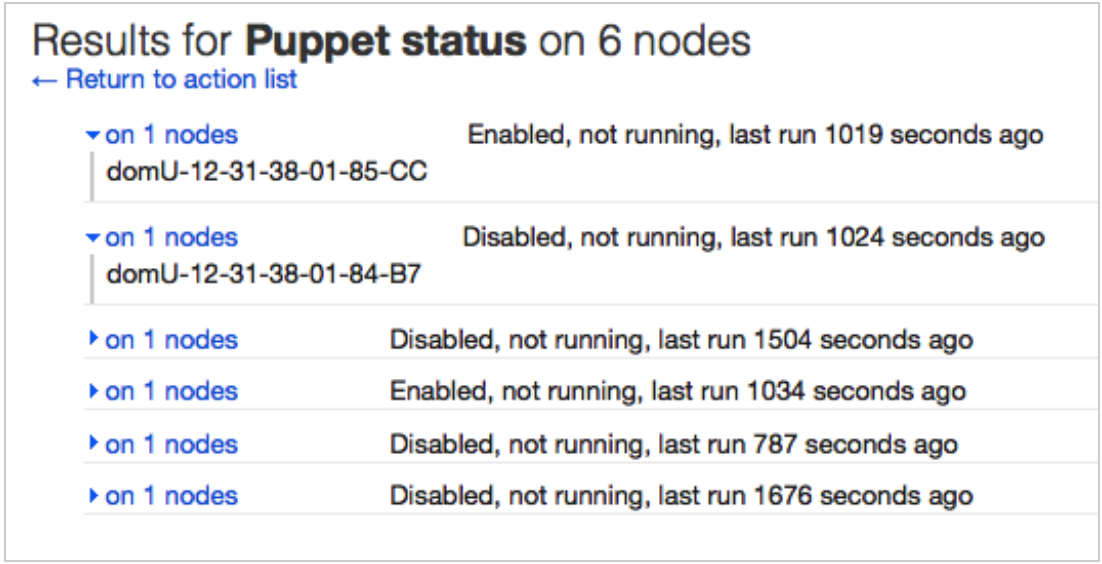


Running an action returns a results view. Results views will attempt to group similar results to reduce noise; click the “on N nodes” links to see which nodes returned a given result.

A collapsed results view:



An expanded results view:



Actions

Run Puppet Once

Use the “runonce” action to make the selected nodes immediately pull and apply their configurations from the puppet master.

Normally, puppet agent pulls configurations at a regular interval (30 minutes, by default). However, when testing new Puppet classes, you’ll probably need to trigger agent runs at irregular times on your test nodes.

Nodes where puppet agent is disabled will ignore this action.

Enable and Disable

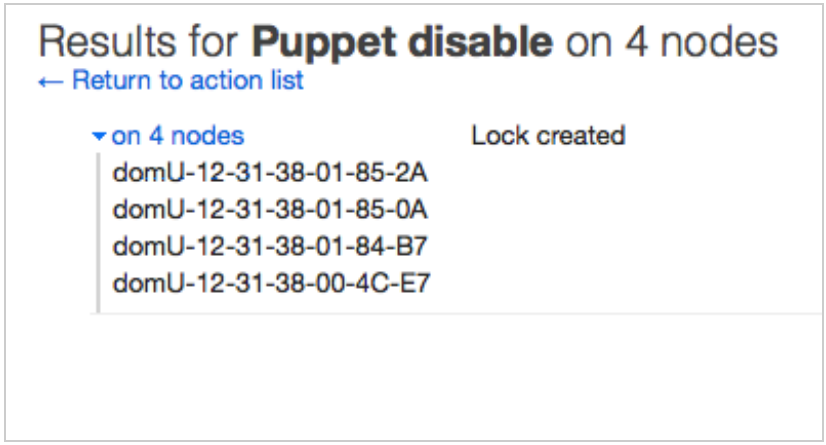
Use the “enable” and “disable” actions to control whether puppet agent does anything. When you disable Puppet on a node, the agent daemon will continue running, but it will not pull configurations from the master.

After a node has been disabled for an hour, it will appear as “unresponsive” in the console’s node views, and will stay that way until it is re-enabled. Disabled nodes will ignore runonce commands from the control puppet tab.

Disabling nodes is great for hedging your bets when you’ve made major changes to an existing Puppet module:

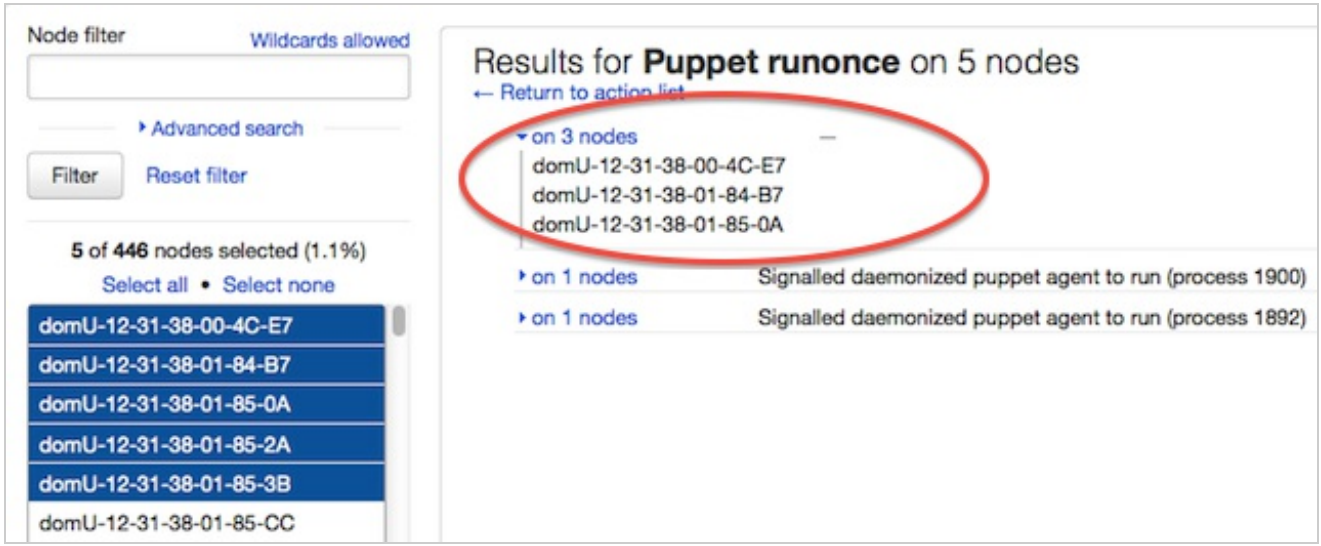
- Filter the node list by Puppet class to find every node that will be affected by the change
- Disable Puppet on all but a few of these nodes
- Run Puppet on the test nodes, and carefully examine the reports to make sure the changes worked
- Re-enable Puppet on the rest of your nodes (or a subset thereof, if you think you need more testing)

A confirmation screen after disabling some nodes:

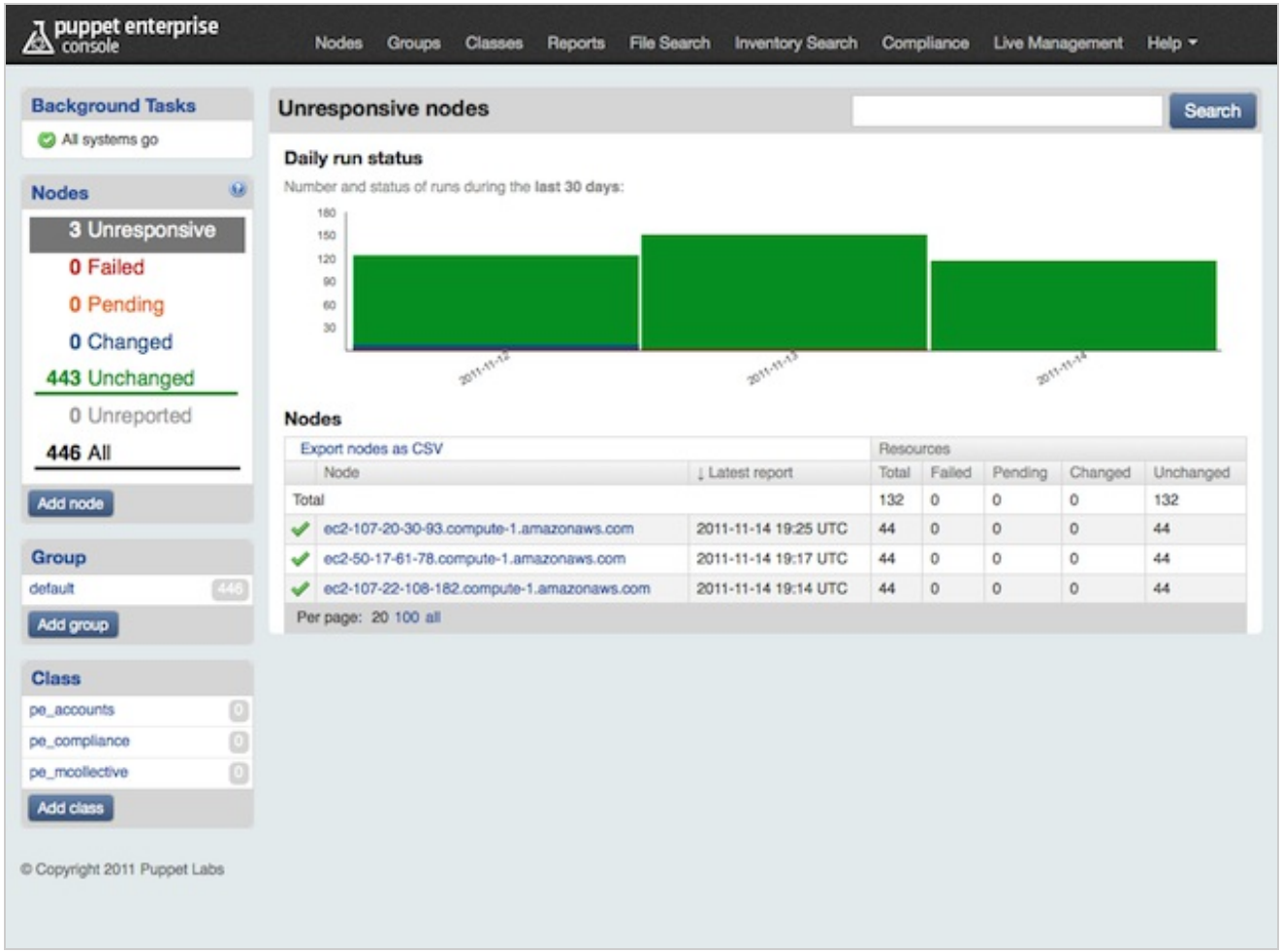


Results after trying to run a mix of enabled and disabled nodes — note the three nodes whose only

response was “—”:



A view of unresponsive nodes in the console (note that node certnames on EC2 don’t necessarily align with hostnames):



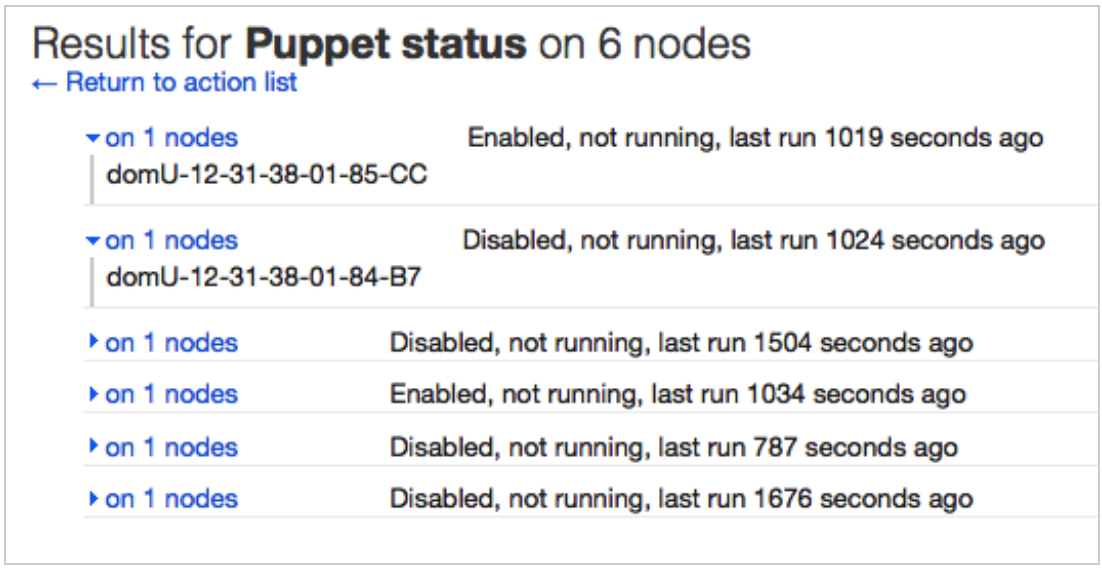
Status

Use the “status” action to check up on puppet agent.

The status action gets three pieces of information from each node:

- Whether Puppet is enabled or disabled
- Whether Puppet is idle (“not running”) or actively applying a configuration (“running”)
- When the last Puppet run occurred

The results of the status action, with a mix of enabled and disabled nodes:

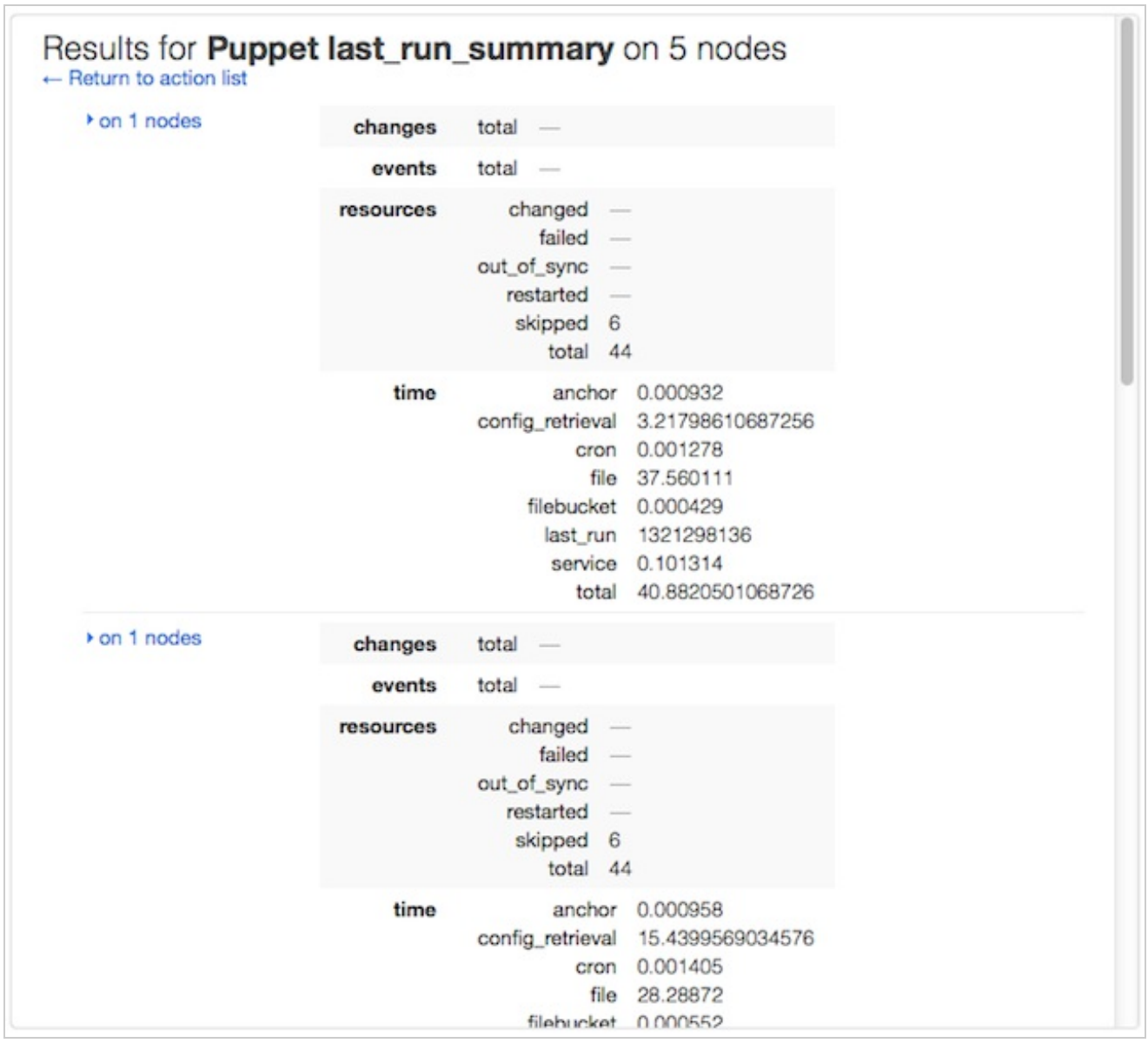


Last Run Summary

Use the “last_run_summary” action for a quick view of what the last Puppet run did.

Usually, you should use the graphs and reports on the console’s node views to investigate previous Puppet runs; they are more detailed, and provide more historical context. However, the overview provided by this action can be useful when combined with live management’s class and fact filtering.

Part of a last run summary results view:



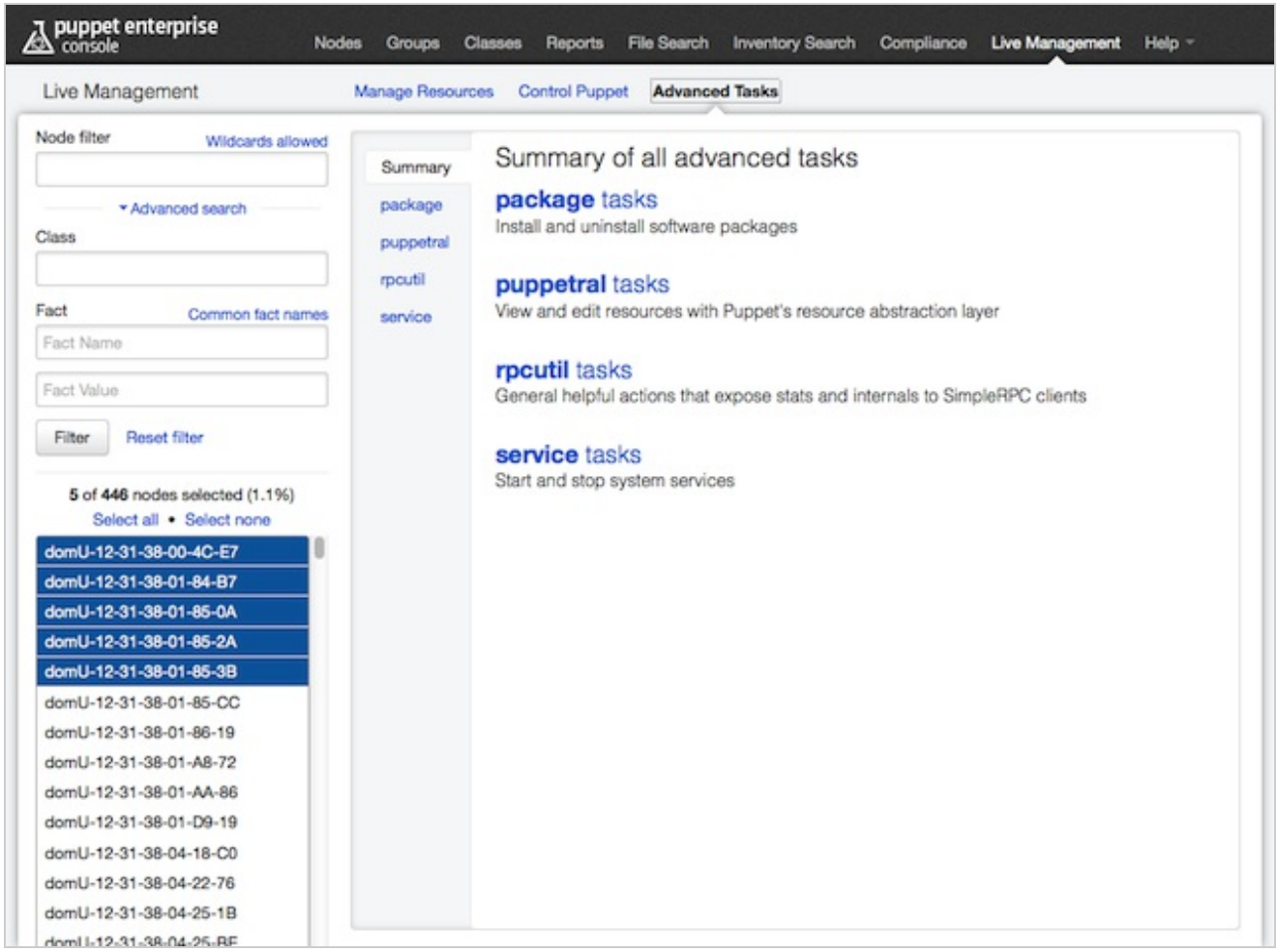
← Console: Live Management: Managing Resources — Index — Console: Live Management: Advanced Tasks →

PE 2.0 » Console » Live Mgmt: Advanced Tasks

← Console: Live Management: Controlling Puppet — Index — Puppet: Overview →

Live Management: Advanced Tasks

Use the advanced tasks tab to invoke actions from any MCollective agent installed on your nodes.



Agents and Actions

This tab is a direct interface to your nodes' MCollective agents. It automatically generates a GUI for every agent installed on your systems, exposing all of their actions through the console. If you install any custom MCollective agents, they'll appear in the advanced tasks tab.

Puppet Enterprise ships with the following MCollective agents:

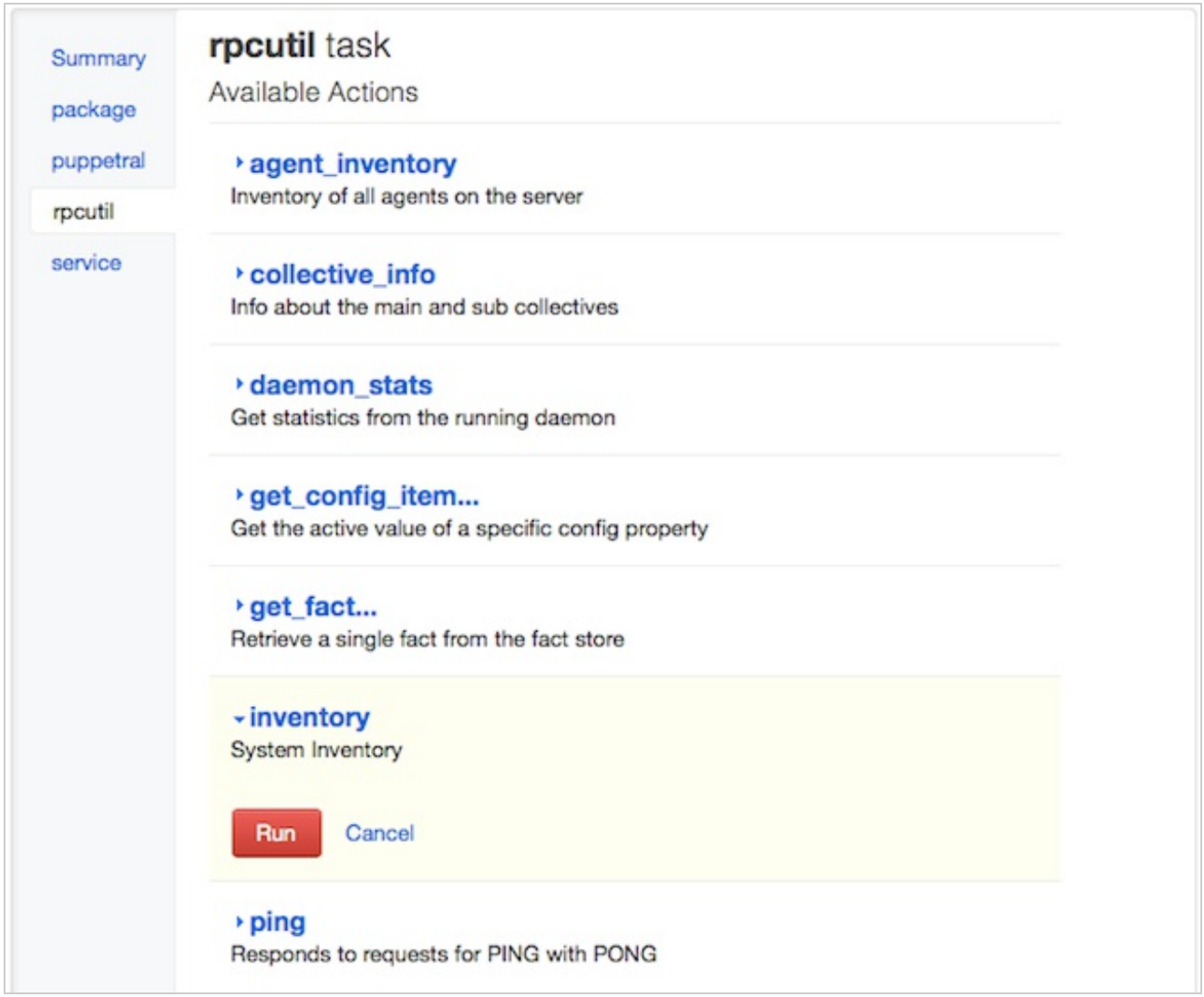
- package — installs and uninstalls software packages.
- rpcutil — performs miscellaneous meta-tasks.
- service — starts and stops services.
- puppetral — the agent used to implement the manage resources tab. When used from the advanced tasks tab, it can only inspect resources, not clone them.

Each agent view includes explanatory text for each action.

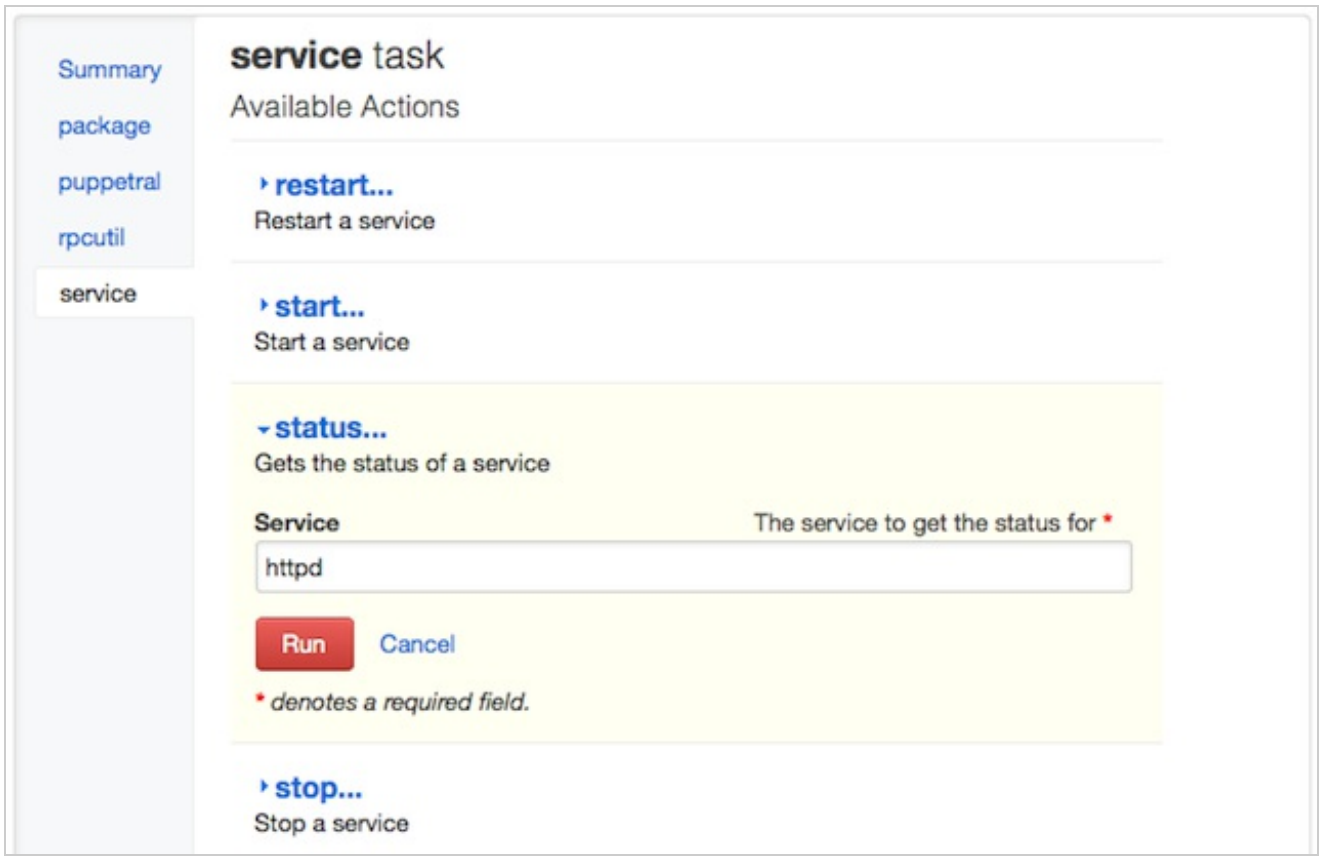
Invoking Actions

Navigate to an agent to see a list of actions. To invoke an action, click its name, enter any required arguments, and confirm with the red “run” button.

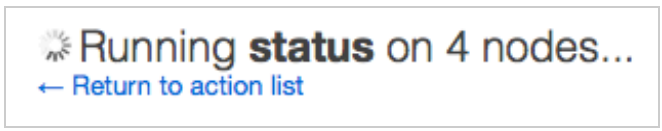
Invoking an action with no arguments:



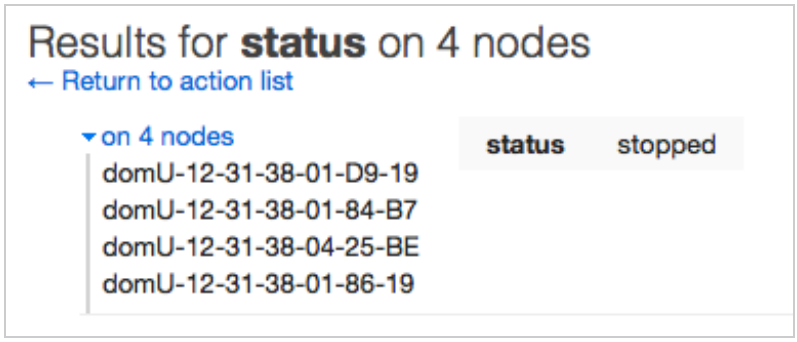
Invoking an action with an argument:



An action in progress:



Results:



← Console: Live Management: Controlling Puppet — Index — Puppet: Overview →

PE 2.0 » Puppet » Overview

← Console: Live Management: Advanced Tasks — Index — Puppet: Your First Module →

Puppet For New Users: Overview

Puppet is divided into two main components:

- The puppet agent daemon runs on every agent node. At regular intervals, it pulls configuration "catalogs" from the puppet master and applies them to the local system. (Puppet catalogs are idempotent: they can be run any number of times and will always converge to the same state.)
- The puppet master service runs on a central server. It decides what configurations each agent node should receive, and compiles nodes' catalogs from "manifests" written by a user.

Where Configurations Come From

Configurations for nodes are compiled from manifests, which are documents written in Puppet's custom language. Manifests declare resources, each of which represents the desired state of some thing (software package, service, user account, file…) on a system. Resources are grouped into classes, and classes are grouped into modules. Modules are structured collections of manifest files where each file contains a single class (or defined type).

The puppet master's collection of modules lives in the `/etc/puppetlabs/puppet/modules` directory. Any class stored in one of the master's modules can be assigned to any of your nodes.

How Configurations are Assigned to Nodes

In Puppet Enterprise, the console controls which classes are assigned to nodes. You can assign classes to nodes individually, or you can collect nodes into groups and assign classes to large numbers of nodes at a time. You can also declare variables ("parameters") that can be read by any of the classes assigned to the node.

When an agent node requests its catalog from the master, the master asks the console which classes and parameters to use, then compiles those classes into the node's catalog.

What Nodes Do With Catalogs

The heart of Puppet is the resource abstraction layer (RAL), which lets the puppet agent turn abstract resource declarations into concrete actions specific to the local system. Once the agent has its catalog of resource declarations, it uses the system's own tools to bring those resources into their desired state.

When New Configurations Take Effect

By default, puppet agent will pull a catalog and run it every 30 minutes (counted from when the agent service started, rather than on the half-hour). You can change this by setting the `runinterval` option in an agent's `/etc/puppetlabs/puppet/puppet.conf` file to a new value. (The `runinterval` is measured in seconds.)

If you need a node or group of nodes to retrieve a new configuration now, use the “Control Puppet” section of the console’s live management page.

← Console: Live Management: Advanced Tasks — Index — Puppet: Your First Module →

PE 2.0 » Puppet » Your First Module

← Puppet: Overview — Index — Puppet: Assigning a Class to a Node →

Puppet For New Users: Your First Module

Where Modules Live

All of your modules belong in `/etc/puppetlabs/puppet/modules`. Puppet Enterprise's built-in modules are stored in `/opt/puppet/share/puppet/modules`.

Module Structure

Modules are directory trees that look like this:

- `core_permissions/` (the module name)
  - `manifests/`
    - `init.pp` (contains the `core_permissions` class)
    - `webserver.pp` (contains the `core_permissions::webserver` class)

File names map to class names in a predictable way: `init.pp` will contain a class with the same name as the module, `<NAME>.pp` will contain a class called `<MODULE NAME>::<NAME>`, and `<NAME>/<OTHER NAME>.pp` will contain `<MODULE NAME>::<NAME>::<OTHER NAME>`. See here for more about the rules for autoloading classes.

Modules can use directories other than `manifests` to do many interesting things, but your first one doesn't need them.

Class Structure

Manifests in a module should contain class definitions. Class definitions in the Puppet language look like this:

```
class core_permissions {
  # Comment
  ... resource declarations ...
}
```

Resource Declarations

Classes should contain resource declarations, which look like this:

```
file {'/etc/fstab':
  ensure => present,
  mode   => 0644,
  owner  => 'root',
  group  => 'root',
```

```
}
```

Resources have:

- A type (“file”)
- A title (“/etc/fstab”)
- …and a series of attribute (“mode”) / value (“0644”) pairs.

Note especially:

- The colon after the title
- The comma after each attribute/value pair

…as resource declarations will break without them.

Every resource type also has one attribute (called the “namevar”) that will default to the title; with the file type, the namevar is “path.” See the core types cheat sheet for a quick reference to the most common resource types.

Your First Complete Module

To put it all together, here’s how to make your first module:

```
# mkdir -p /etc/puppetlabs/puppet/modules/core_permissions/manifests
# vi /etc/puppetlabs/puppet/modules/core_permissions/manifests/init.pp
```

Paste the following code into the `init.pp` file:

```
class core_permissions {
  file {'/etc/fstab':
    ensure => present,
    mode   => 0644,
    owner  => 'root',
    group  => 'root',
  }

  file {'/etc/passwd':
    ensure => present,
    mode   => 0644,
    owner  => 'root',
    group  => 'root',
  }

  file {'/etc/crontab':
    ensure => present,
    mode   => 0644,
    owner  => 'root',
    group  => 'root',
  }
```

```
    }
```

This creates a class called `core_permissions`, which manages the ownership and mode of three important files. On any that node that has this class applied to it, an accidental or malicious change in these files' permissions would be reverted to the specified state within the next half-hour, and a report of the change would appear on that node's page in the console.

Remember that creating the module won't automatically assign the class to any nodes — it just makes the class available.

← Puppet: Overview — Index — Puppet: Assigning a Class to a Node →

PE 2.0 » Puppet » Assigning a Class to a Node

← Puppet: Your First Module — Index — Puppet: Next Steps →

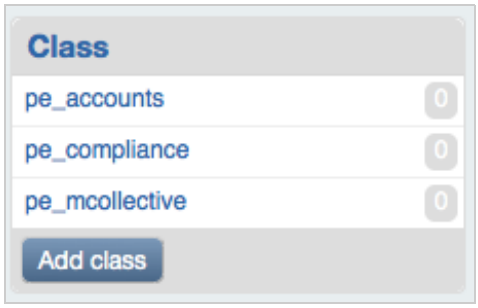
Puppet For New Users: Assigning a Class to a Node

In Puppet Enterprise, you assign classes to nodes using the console.

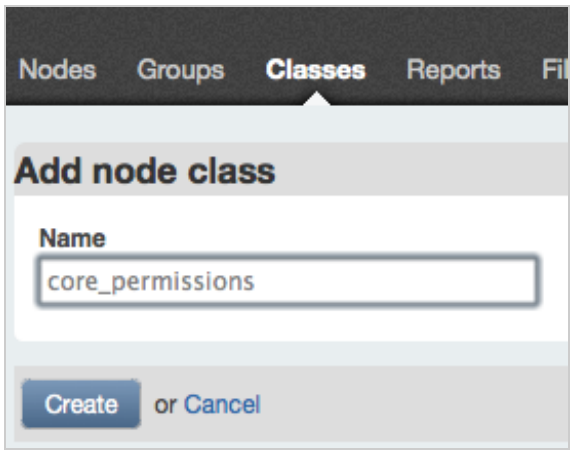
Adding a Class to the Console

The console doesn't automatically load classes from the puppet master's modules, so you must tell the console about a new class before you can use it. Use the "Add class" button in the console's sidebar.

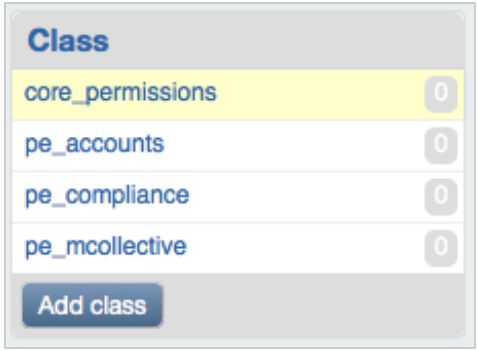
The "Add class" button:



Typing a class name:



Done:

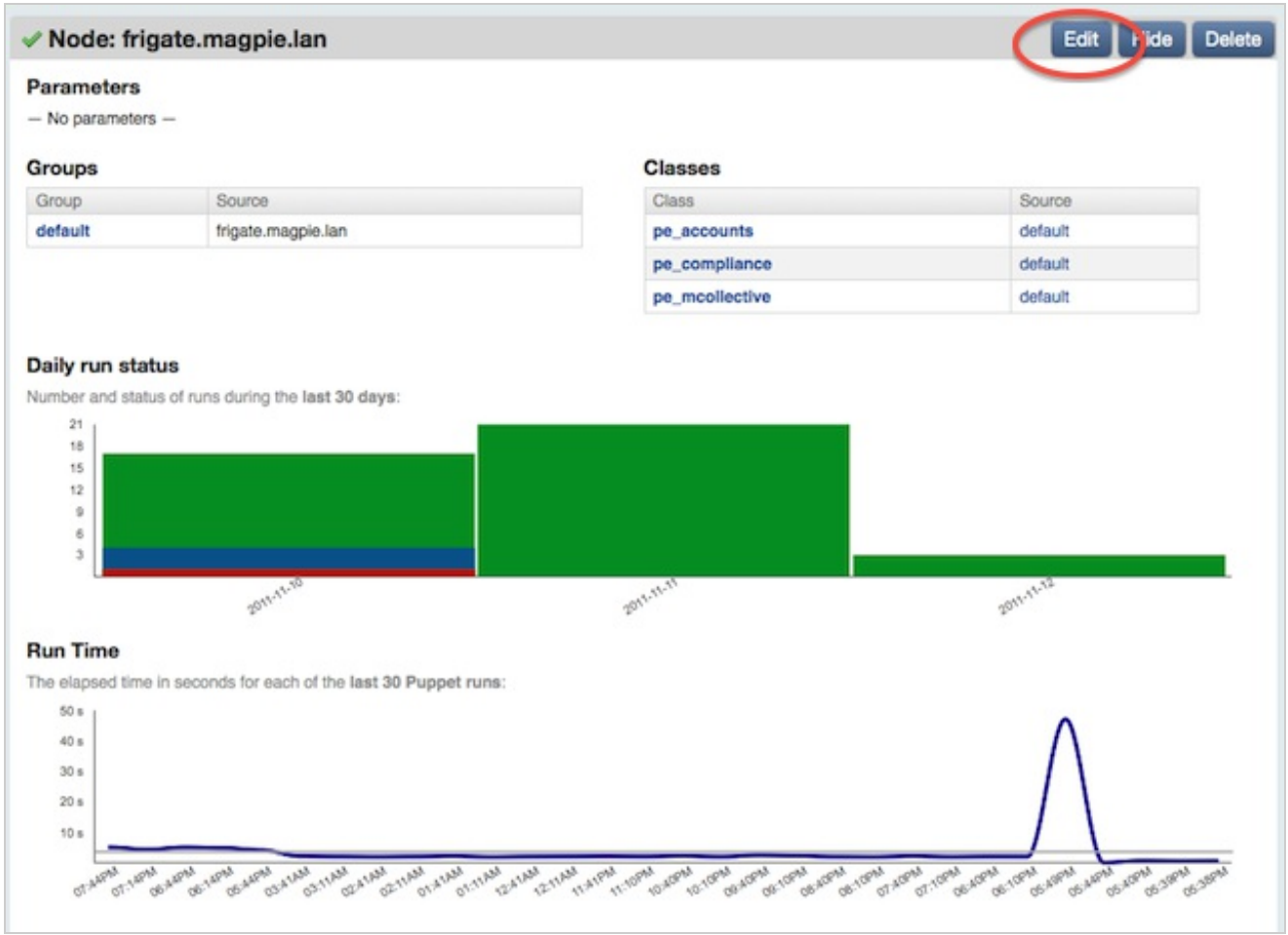


Assigning a Class to a Single Node

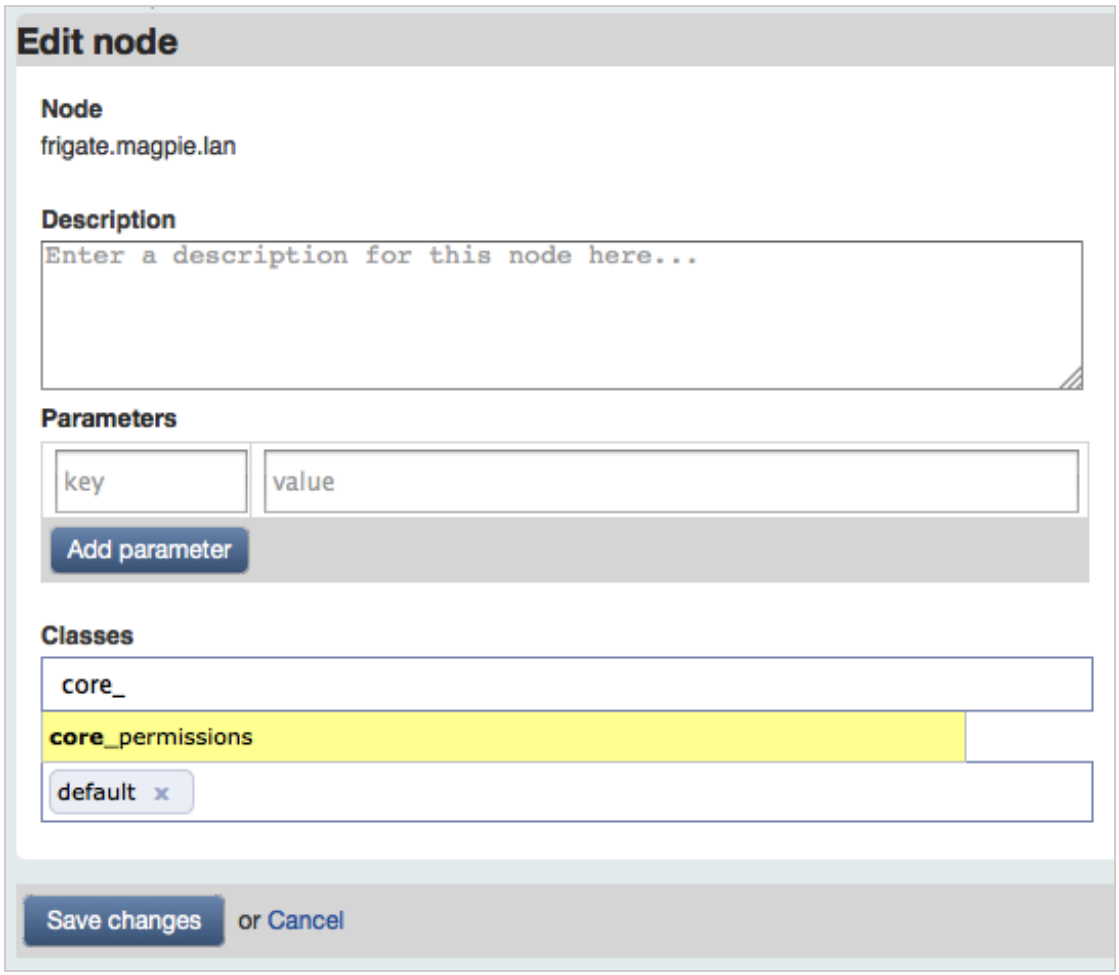
To assign the new class to one node at a time:

- Go to that node's page (by finding and clicking its name in one of the console's lists of nodes)
- Click "Edit"
- Start typing the name of the class in its "Classes" field, then select the class you want from the auto-completion list
- Save your changes

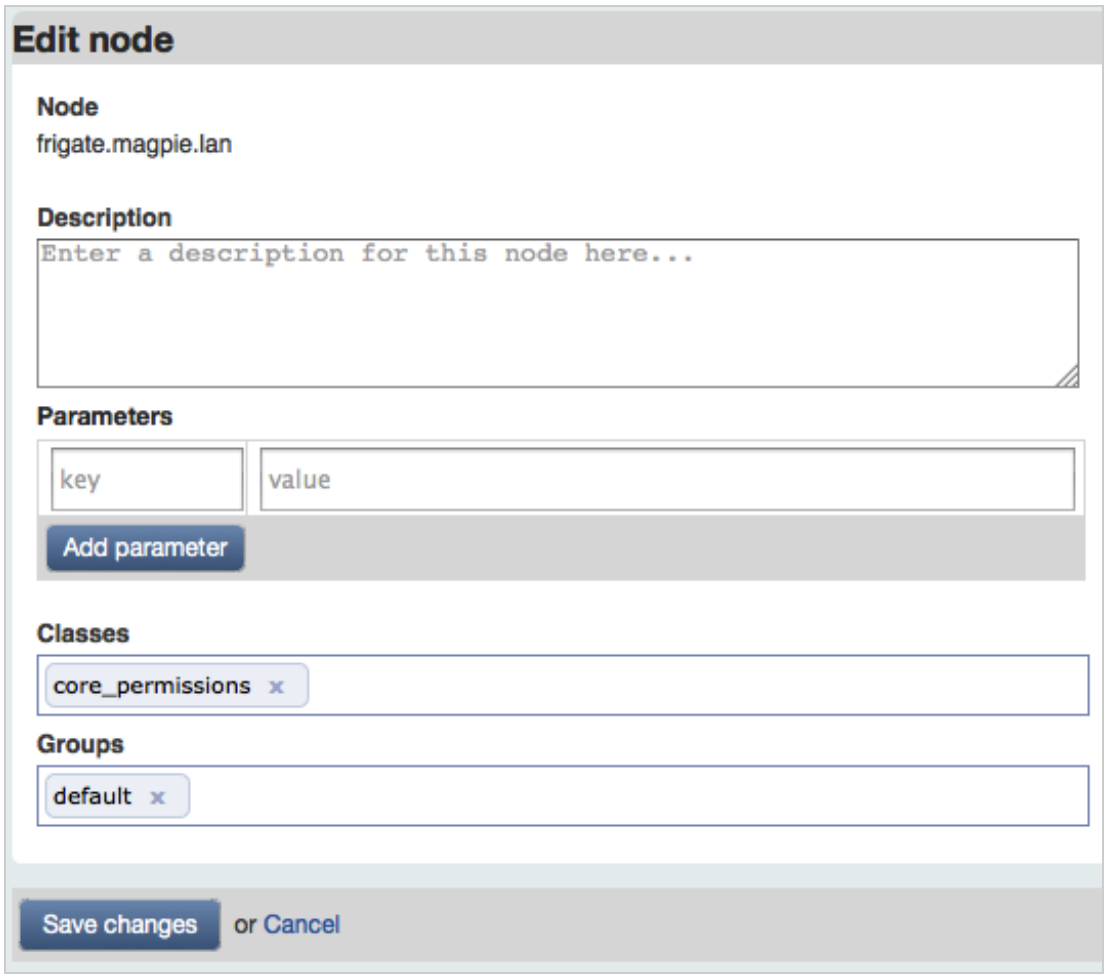
The edit button:



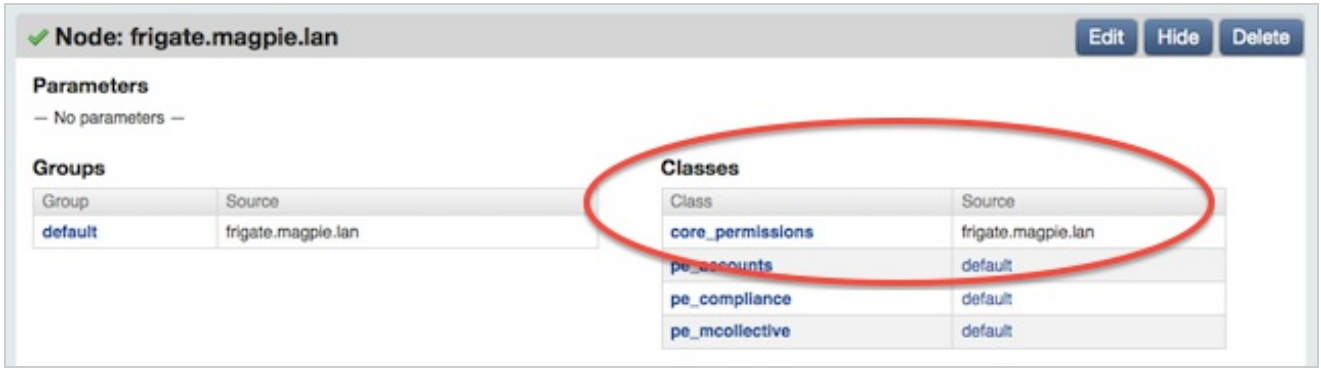
Typing a class name:



A confirmed class name:□



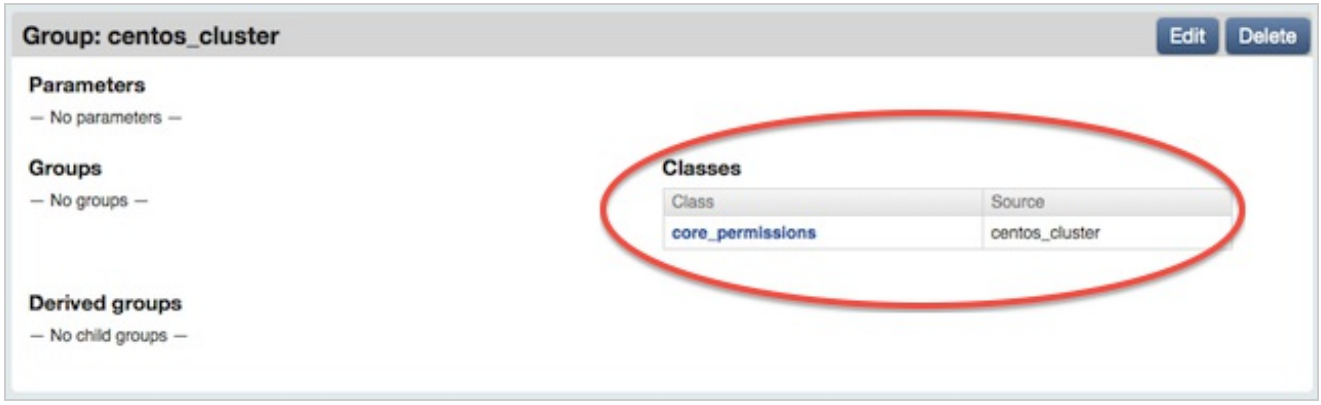
Done:



Assigning a Class to a Group

Assigning a class to a group of nodes is nearly identical: go to that group's page, and use the edit button and classes field as described above.□

After adding a class to a group:

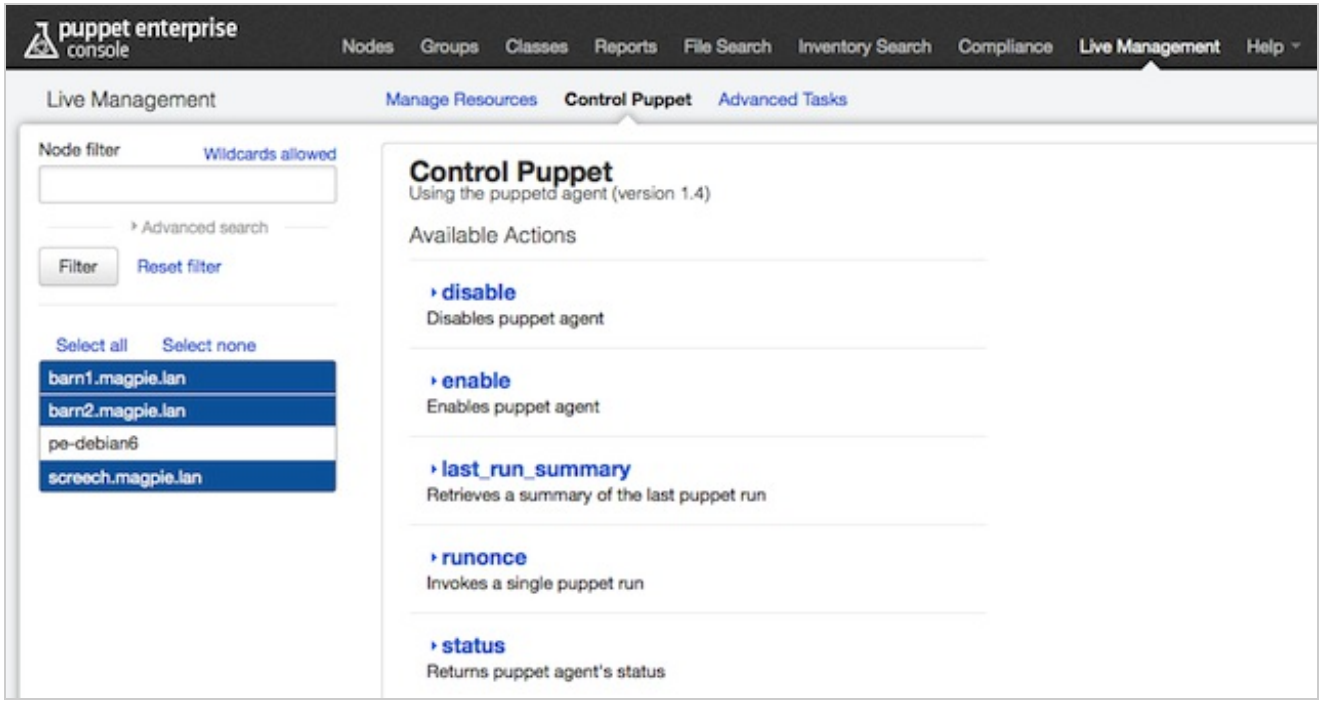


Making a Node Pull its Configuration

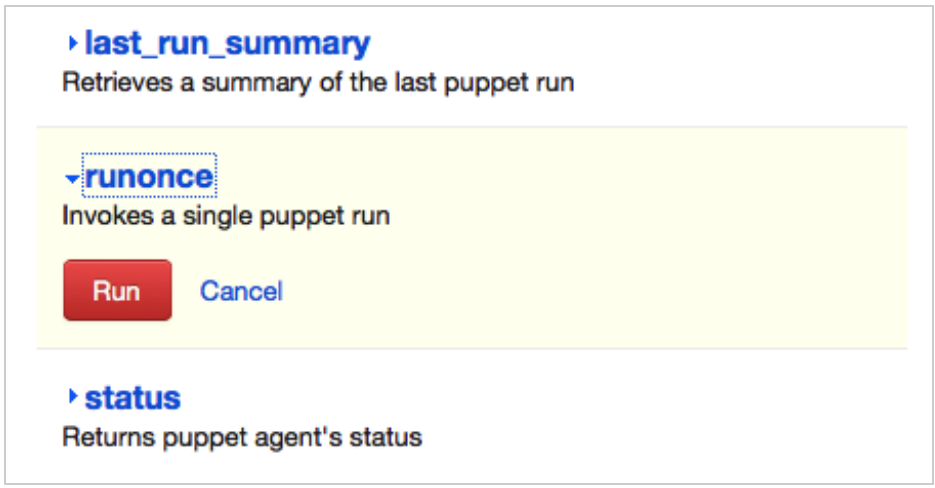
Nodes with your class will pull and apply their configurations within the next half hour. But if you want to make them run immediately, you can use the console's live management page to control puppet agent.

- Navigate to the live management page, then navigate to the "Control Puppet" section
- In the sidebar, click "select none" and then re-select the handful of nodes that need to run
- Click the "runonce" action, then confirm with the red "Run" button.

Preparing to trigger three runs:

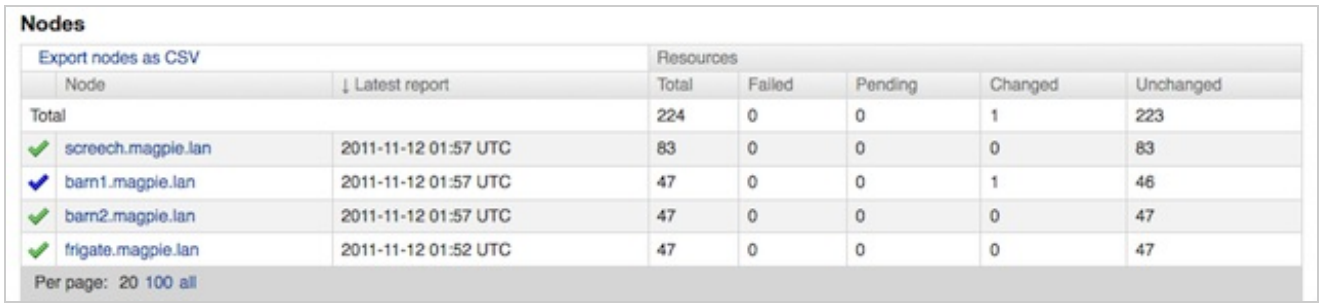


Confirming the runonce action:



Viewing the Results of a Run

A few minutes after triggering a run, the selected nodes should be the most recent nodes appearing in the node list. In our case, we can see that this run made changes to one node, which has a blue checkmark by its name:



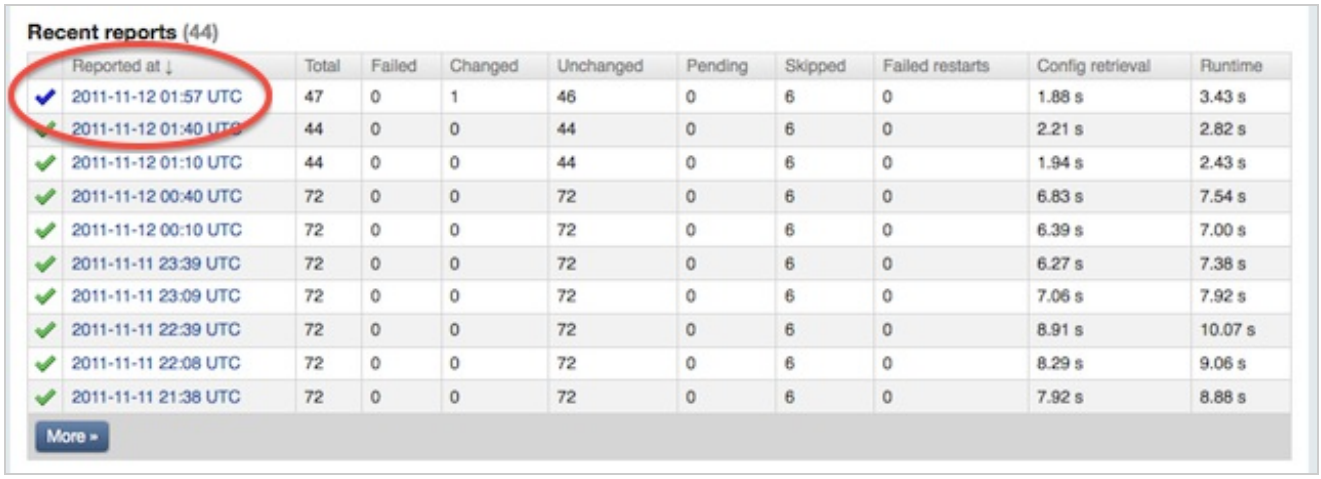
Nodes

Export nodes as CSV

Node	↓ Latest report	Resources Total	Failed	Pending	Changed	Unchanged
Total		224	0	0	1	223
screech.magpie.lan	2011-11-12 01:57 UTC	83	0	0	0	83
barn1.magpie.lan	2011-11-12 01:57 UTC	47	0	0	1	46
barn2.magpie.lan	2011-11-12 01:57 UTC	47	0	0	0	47
frigate.magpie.lan	2011-11-12 01:52 UTC	47	0	0	0	47

Per page: 20 100 all

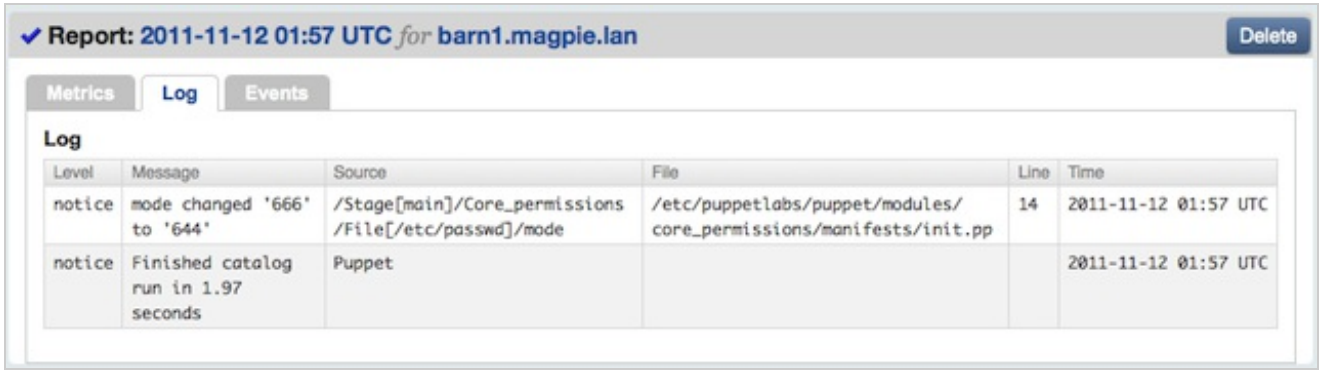
If we go to that node's page, we can go to the most recent report and view what happened in the "log" tab:



Recent reports (44)

Reported at ↓	Total	Failed	Changed	Unchanged	Pending	Skipped	Failed restarts	Config retrieval	Runtime
2011-11-12 01:57 UTC	47	0	1	46	0	6	0	1.88 s	3.43 s
2011-11-12 01:40 UTC	44	0	0	44	0	6	0	2.21 s	2.82 s
2011-11-12 01:10 UTC	44	0	0	44	0	6	0	1.94 s	2.43 s
2011-11-12 00:40 UTC	72	0	0	72	0	6	0	6.83 s	7.54 s
2011-11-12 00:10 UTC	72	0	0	72	0	6	0	6.39 s	7.00 s
2011-11-11 23:39 UTC	72	0	0	72	0	6	0	6.27 s	7.38 s
2011-11-11 23:09 UTC	72	0	0	72	0	6	0	7.06 s	7.92 s
2011-11-11 22:39 UTC	72	0	0	72	0	6	0	8.91 s	10.07 s
2011-11-11 22:08 UTC	72	0	0	72	0	6	0	8.29 s	9.06 s
2011-11-11 21:38 UTC	72	0	0	72	0	6	0	7.92 s	8.88 s

More »



For some reason, `/etc/passwd` had a permissions mode of 0666, which meant anyone could write to it! Puppet corrected that, and it now has the proper mode of 0644.

← Puppet: Your First Module — Index — Puppet: Next Steps →

PE 2.0 » Puppet » Next Steps

← Puppet: Assigning a Class to a Node — Index — Orchestration: Overview →

Puppet For New Users: Next Steps

More Resources

Learning Puppet

The preceding chapters were sprinkled with links to Learning Puppet, our primary resource for new Puppet users. To learn more about how to write classes and modules, how to arrange resources into useful combinations, and how Puppet sees the world, start at the beginning. You can download the Learning Puppet VM to start with, or you can simply use the test nodes you've already installed Puppet Enterprise on. Learning Puppet is a work in progress and is being actively improved.

More About Resource Types

For a fast, printable reference to the most common resource types, get the Core Types Cheat Sheet. For more detailed info about Puppet's resource types, see the type reference for your version of Puppet.

More About the Puppet Language

For more complete details about the Puppet language, see the language guide.

More About the Console

For more on Puppet Enterprise's console, see the console section of this user's guide.

← Puppet: Assigning a Class to a Node — Index — Orchestration: Overview →

PE 2.0 » Orchestration » Overview

← Puppet: Next Steps — Index — Orchestration: Usage and Examples →

Orchestration for New PE Users: Overview

What is Orchestration?

Orchestration means invoking actions in parallel across any number of nodes at once.

PE's orchestration features are built on the MCollective framework, which consists of the following components:

- Client interfaces can issue orchestration commands to some or all of your nodes. The console's live management tools are one client interface, and the `mco` command line tool is another.
- Orchestration agents are plugins that live on agent nodes and provide orchestration actions.
- The MCollective service runs on every agent node and listens for orchestration commands. If a command is legit, relevant to the node, and for a supported action, the service will trigger the action and send back results.
- The message broker is a central server that routes orchestration messages between client interfaces and nodes running the MCollective service. (PE's ActiveMQ message server runs on the puppet master node.)

Orchestration isn't SSH

Orchestration isn't for running arbitrary code on nodes. Instead, each node has a collection of actions available. Actions are distributed in plugins, and you can extend PE's orchestration features by downloading or writing new orchestration agents and distributing them with Puppet.

Live Management is Orchestration

The console's live management page offers a convenient graphical interface for orchestration tasks, such as browsing and cloning resources across nodes. See the live management chapters of this user's guide for more details.

Orchestration is Also Scriptable

In addition to live management's interactive interface, PE includes command-line tools that let you script and automate orchestration tasks (or just run them from the comfort of your terminal).

Changes Since PE 1.2

PE's orchestration features have been changed and improved since they were introduced in version 1.2.

- Orchestration is enabled by default for all PE nodes.
- Orchestration tasks can now be invoked directly from the console, with the "advanced tasks" tab of the live management page. PE's orchestration framework also powers the other live management features.
- The `mco` user account on the puppet master is gone, in favor of a new `peadmin` user. This user

can still invoke orchestration tasks across your nodes, but it will also gain more general purpose capabilities in future versions.

- PE now includes the `puppetral` plugin, which lets you use Puppet's Resource Abstraction Layer (RAL) in orchestration tasks.
- For performance reasons, the default message security scheme has changed from AES to PSK.
- The network connection over which messages are sent is now encrypted using SSL.

Security

All network traffic for orchestration is encrypted with SSL (without host verification). In addition, all orchestration messages are authenticated using a randomly generated pre-shared key (PSK).

- If necessary, you can change the password used as the pre-shared key.
- You can also change the authentication method to use an AES key pair instead of a pre-shared key. (Note that this can potentially affect performance with large numbers of nodes.)

Network Traffic

Nodes send orchestration messages over TCP port 61613 to the ActiveMQ server, which runs on the puppet master node. See the notes on firewall configuration in the "Preparing to Install" chapter of this guide for more details about PE's network traffic.

← Puppet: Next Steps — Index — Orchestration: Usage and Examples →

PE 2.0 » Orchestration » Usage

← Orchestration: Overview — Index — Cloud Provisioning: Overview →

Orchestration for New PE Users: Usage and Examples

Running Actions

Orchestration actions are grouped and distributed as MCollective plugins. In the default installation of Puppet Enterprise, you can run any orchestration action from any plugin while logged in as the `peadmin` user on the puppet master node.

To open a shell as the `peadmin` user, run:

```
# sudo -i -u peadmin
```

To run orchestration actions, use the `mco` command:

```
$ mco <PLUGIN> <FILTER> <ACTION> <ARGUMENT> <OPTIONS>
```

Available Actions

The following orchestration actions are available in PE 2.0:

- `rpcutil` plugin
  - `find` action returns a list of all nodes matching a search filter
  - `ping` action returns a list of all nodes and their latencies
  - `inventory` action returns a list of Facts, Classes, and other information from all nodes
  - this plugin's actions are exposed via higher-level wrappers, such as the mco ping command
- `puppetral` plugin
  - `find` action returns a Puppet resource of a given type and title and its variations across all nodes
  - `search` action returns all Puppet resources of a given type across all nodes
  - `create` action creates a given resource across all nodes
    - creating an exec resource allows for arbitrary management of nodes
- `puppetd` plugin
  - `enable` and `disable` actions enable and disable puppet agent on a node or nodes
  - `runonce` action initiates a puppet agent run on all nodes
  - `last_run_summary` action retrieves the most recent Puppet run summary from all nodes

- `status` action returns puppet agent's run status on all nodes

- `service` plugin
  - `start, stop, restart,` and `status` actions allow direct management of services across the deployment

- `package` plugin
  - `install, purge, checkupdate, update,` and `status` actions work through the system package manager to query and ensure the state of software packages across the deployment

- `controller` plugin
  - `stats` action returns cumulative statistics about MCollective messages passed between nodes
  - `reload_agent` action refreshes from disk the code for a specific plugin
  - `reload_agents` action refreshes from disk the code for all plugins
  - `exit` action removes nodes from the MCollective network

Filtering Nodes

Most orchestration actions in Puppet Enterprise 2.0 can be executed on a set of nodes determined by meta-data about the deployment. This filtering provides a much more convenient way to manage nodes than the traditional approach of using host names or fully qualified domain names to identify and access machines. Node sets can be created by filtering based on Facter facts, Puppet classes, and host names. Filters can be specified by passing options to the mco command. For example:

```
$ mco find --with-fact osfamily=RedHat
```

This command limits the find action to only return nodes who have a fact named osfamily with a value of RedHat. Filter options are case sensitive and support regular expression syntax.

Examples

Ping

The ping action of the rpcutil plugin is wrapped to be available at a higher level as the mco ping command. This command returns a list of all the nodes and their network latencies. In a typical Puppet Enterprise deployment, latencies for issuing an orchestration action are less than half a second:

```
peadmin@puppetmaster:~$ mco ping
puppetmaster.example.com                time=135.94 ms
agent.example.com                       time=136.55 ms
---- ping statistics ----
```

```
2 replies max: 136.55 min: 135.94 avg: 136.25
```

Find

The find action of the rpcutil plugin is wrapped to be available at a higher level as the mco find command. This command returns a list of all the nodes in the network:

```
peadmin@puppetmaster:~$ mco find
puppetmaster.example.com
agent.example.com
```

Inventory

The inventory action of the rpcutil plugin is wrapped to be available at a higher level as the mco inventory command. This command returns facts and classes, among other information:

```
peadmin@puppetmaster:~$ mco inventory agent.example.com
Inventory for agent.example.com:
 Server Statistics:
   Version: 1.2.1
   Start Time: Mon Nov 14 15:25:33 -0800 2011
   Config File: /etc/puppetlabs/mcollective/server.cfg
   Collectives: mcollective
   Main Collective: mcollective
   Process ID: 5553
   Total Messages: 86
   Messages Passed Filters: 74
   Messages Filtered: 12
   Replies Sent: 73
   Total Processor Time: 1.34 seconds
   System Time: 0.59 seconds
 Agents:
   discovery       package         puppetd
   puppetral       rpcutil         service
 Configuration Management Classes:
   default                        helloworld
   helloworld                     pe_accounts
   pe_accounts                    pe_accounts::data
   pe_accounts::groups            pe_compliance
   pe_compliance                  pe_compliance::agent
   pe_mcollective                 pe_mcollective
   pe_mcollective::metadata       pe_mcollective::plugins
   settings
Facts:
   architecture => i386
   augeasversion => 0.7.2
   ...
```

Puppetd

The puppetd plugin orchestrates Puppet itself across the deployment. This plugin is capable of

kicking off Puppet configuration runs on each node in the deployment all at the same time, on a subset of the deployment using filtering, or using a maximum concurrency level. The maximum concurrency feature is particularly noteworthy as it allows the execution of a configuration run on all nodes in the population, but sets an upper limit on the number of nodes that are performing their run at any given time, thus mitigating the network load and resource demands on the puppet master. This example performs a configuration run on all nodes, but only one node at a time:

```
peadmin@puppetmaster:~$ mco puppetd runall 1
Tue Nov 15 11:17:11 -0800 2011> Running all machines with a concurrency of 1
Tue Nov 15 11:17:11 -0800 2011> Discovering hosts to run
Tue Nov 15 11:17:13 -0800 2011> Found 2 hosts
Tue Nov 15 11:17:13 -0800 2011> Running agent.example.com, concurrency is 0
Tue Nov 15 11:17:14 -0800 2011> agent.example.com schedule status: OK
Tue Nov 15 11:17:15 -0800 2011> Currently 1 nodes running; waiting
Tue Nov 15 11:17:21 -0800 2011> Running puppetmaster.example.com, concurrency
is 0
Tue Nov 15 11:17:22 -0800 2011> puppetmaster.example.com schedule status: OK
Finished processing 1 / 1 hosts in 98.98 ms
```

Note, the “1 / 1 hosts” is an indication that one host was actually performing the orchestration of the Puppet configuration run. This does not mean only one host in the deployment performed the configuration run. The message “Found 2 hosts” indicates that two nodes carried out this action.

Once finished with the Puppet configuration run, the status action can be used to see the last Puppet configuration run for each node in the deployment.

```
peadmin@puppetmaster:~$ mco puppetd status
[ ============================================> ] 2 / 2
agent.example.com  Enabled, not running, last run 10 seconds ago
puppetmaster.example.com  Enabled, not running, last run 2 seconds ago

Finished processing 2 / 2 hosts in 105.29 ms
```

PuppetRAL

The puppetral plugin provides a command line tool to work with any resource Puppet is capable of managing. A common use case for the puppetral plugin is to execute arbitrary commands on the deployment using the exec resource. This example creates a new file on all nodes in the deployment:

```
peadmin@puppetmaster:~$ mco rpc puppetral \
create type=exec title="/bin/bash -c 'echo Hello > /tmp/hello'"
Determining the amount of hosts matching filter for 2 seconds .... 2

[ ============================================> ] 2 / 2

agent.example.com
Status: Resource was created
```

```
Resource:
  {"exported"=>false,
   "title"=>"/bin/bash -c 'echo Hello > /tmp/hello'",
   "parameters"=>{:returns=>:notrun},
   "tags"=>["exec"],
   "type"=>"Exec"}

 puppetmaster.example.com
 Status: Resource was created
 Resource:
   {"exported"=>false,
    "title"=>"/bin/bash -c 'echo Hello > /tmp/hello'",
    "parameters"=>{:returns=>:notrun},
    "tags"=>["exec"],
    "type"=>"Exec"}

 Finished processing 2 / 2 hosts in 249.21 ms
```

That this file was created can be verified with the find action:

```
 peadmin@puppetmaster:~$ mco rpc puppetral find type=file title=/tmp/hello

 Determining the amount of hosts matching filter for 2 seconds .... 2

 [ ============================================> ] 2 / 2

 agent.example.com
   exported: false
   managed: false
   title: /tmp/hello
   parameters:
     {:ctime=>Tue Nov 15 11:32:10 -0800 2011,
      :type=>"file",
      :ensure=>:file,
      :group=>0,
      :owner=>0,
      :mtime=>Tue Nov 15 11:32:10 -0800 2011,
      :mode=>"644",
      :content=>"{md5}09f7e02f1290be211da707a266f153b3"}
  tags: ["file"]
  type: File

 puppetmaster.example.com
   managed: false
   exported: false
   title: /tmp/hello
   parameters:
     {:ctime=>Tue Nov 15 11:32:10 -0800 2011,
      :type=>"file",
      :group=>0,
      :ensure=>:file,
      :owner=>0,
      :mtime=>Tue Nov 15 11:32:10 -0800 2011,
      :mode=>"644",
      :content=>"{md5}09f7e02f1290be211da707a266f153b3"}
```

```
   tags: ["file"]
   type: File

 Finished processing 2 / 2 hosts in 166.12 ms
```

The puppetral plugin can manage any resource Puppet itself is capable of managing. In particular, user, group, host, and package resources are exposed directly in the console's live management page.

Service

Puppet agent will automatically restart a service it manages when related resources are changed, but sometimes services need to be restarted outside of a normal Puppet configuration run. The mco service command can be used in these cases. This example restarts the SSH daemon on all nodes running RedHat:

```
peadmin@puppetmaster:~$ mco service sshd restart -W osfamily=RedHat
[ ============================================> ] 2/ 2
---- service summary ----
Nodes: 2 / 2
Statuses: started=2
Elapsed Time: 1.45 s
```

The status of the restarted services can be found with:

```
peadmin@puppetmaster:~$ mco service sshd status -W osfamily=RedHat
[ ============================================> ] 2 / 2
puppetmaster.example.com              status=running
agent.example.com                     status=running
---- service summary ----
Nodes: 2 / 2
Statuses: started=2
Elapsed Time: 0.26 s
```

Package

Similar to the service plugin, the package manages software packages outside of a normal Puppet configuration run. The following is the trimmed output of running the checkupdates action:

```
peadmin@puppetmaster:~$ mco rpc package checkupdates -W fqdn=agent.example.com
Determining the amount of hosts matching filter for 2 seconds .... 1
 * [ ============================================================> ] 1 / 1
agent.example.com
   Outdated Packages: [{:package=>"glibc.i686", :version=>"2.5-65.el5_7.1",
:repo=>"base_local"},
                       {:package=>"nscd.i386", :version=>"2.5-65.el5_7.1",
:repo=>"base_local"},
                       {:package=>"ntp.i386",
                        :version=>"4.2.2p1-15.el5.centos.1",
```

```
                          :repo=>"base_local"},
                         {:package=>"openssh.i386", :version=>"4.3p2-72.el5_7.5",
:repo=>"base_local"},
                         {:package=>"openssh-clients.i386",
                          :version=>"4.3p2-72.el5_7.5",
                          :repo=>"base_local"},
                         {:package=>"openssh-server.i386",
                          :version=>"4.3p2-72.el5_7.5",
                          :repo=>"base_local"}]
               Output:
glibc.i686                        2.5-65.el5_7.1
base_local
nscd.i386                         2.5-65.el5_7.1
base_local
ntp.i386                          4.2.2p1-15.el5.centos.1
base_local
openssh.i386                      4.3p2-72.el5_7.5
base_local
openssh-clients.i386              4.3p2-72.el5_7.5
base_local
openssh-server.i386               4.3p2-72.el5_7.5
base_local
           Exit Code: 100
     Package Manager: yum

Finished processing 1 / 1 hosts in 409.14 ms
```

Controller

The mco controller command manages the underlying MCollective infrastructure. This can be used to load plugins obtained outside of Puppet Enterprise or custom written for the deployment. This example reloads all plugins across the Puppet Enterprise deployment:

```
peadmin@puppetmaster:~$ mco controller reload_agents
Determining the amount of hosts matching filter for 2 seconds .... 2
agent.example.com> reloaded all agents
puppetmaster.example.com> reloaded all agents
Finished processing 2 / 2 hosts in 137.86 ms
```

← Orchestration: Overview — Index — Cloud Provisioning: Overview →

PE 2.0 » Cloud Provisioning » Overview

← Orchestration: Usage and Examples — Index — Cloud Provisioning: Configuring and Troubleshooting →

A Cloud Provisioning Overview

Puppet Enterprise ships with command-line tools for provisioning new nodes. You can use these tools to:

- Create and destroy virtual machine instances on VMware vSphere and Amazon EC2
- Classify new nodes (virtual or physical) in the console
- Automatically install and configure PE on new nodes (virtual or physical)

When used together, these tools provide a quick and efficient workflow for adding nodes to your Puppet Enterprise environment.

See the chapters on VMware and AWS provisioning for details about creating and destroying virtual machines. After that, the chapter on classifying nodes and installing PE covers actions that work on any new machine, virtual or physical.

Tools

PE's provisioning tools are based around the `node`, `node_vmware`, and `node_aws` subcommands. Each of these subcommands have a selection of available actions (such as `list` and `start`). You can get information about a subcommand or its actions with the `puppet help` and `puppet man` commands.

The VMware and AWS subcommands are only used for provisioning, but `node` is a pre-existing Puppet subcommand with several provisioning actions added to it. The `node` actions used in the provisioning process are:

- `classify`
- `init`
- `install`

You may also find the `clean` action useful when decommissioning nodes.

The VMware and AWS provisioning tools are powered by Fog, the Ruby cloud services library. Fog is automatically installed on any machine receiving the cloud provisioner role.

Prerequisites

The cloud provisioning tools ship with Puppet Enterprise 2.0 and later.

Services

The following services and credentials are required:

For VMware you will need:

- VMware vSphere 4.0 and later
- VMware vCenter

For Amazon Web Services you will need:

- An existing Amazon account with support for EC2

Installing

Cloud provisioning can be installed on any puppet master or agent node.

The Puppet Enterprise installer and upgrader ask whether to install cloud provisioning during installation; answer 'yes' to enable cloud provisioning actions on a given node.

If you're using an answer file to install Puppet Enterprise, this capability can be installed by setting the `q_puppet_cloud_install` option to `y`.

```
q_puppet_cloud_install=y
```

← Orchestration: Usage and Examples — Index — Cloud Provisioning: Configuring and Troubleshooting →

PE 2.0 » Cloud Provisioning » Configuring and Troubleshooting

---

← Cloud Provisioning: Overview — Index — Cloud Provisioning: Provisioning with VMware →

Configuring and Troubleshooting Cloud Provisioning

Configuring

To create new virtual machines with Puppet Enterprise, you'll need to first configure the services you'll be using.

First, create a file called `.fog` in the home directory of the user who will be provisioning new nodes.

```
$ touch ~/.fog
```

This is the configuration file for Fog, the cloud abstraction library that powers PE's provisioning tools. When filled, it will be a YAML hash containing the locations of your cloud services and the credentials necessary to control them. For example:

```
:default:
  :vsphere_server: vc01.example.com
  :vsphere_username: cloudprovisioner
  :vsphere_password: abc123
  :vsphere_expected_pubkey_hash:
431dd5d0412aab11b14178290d9fcc5acb041d37f90f36f888de0cebfffff0a8
  :aws_access_key_id: AKIAIISJV5TZ3FPWU3TA
  :aws_secret_access_key: ABCDEFGHIJKLMNOP1234556/s
```

Continue reading for explanations of how to find these credentials.

Adding VMware Credentials

To connect to a VMware vSphere server, you must put the following information in your `~/.fog` file:

`:vsphere_server`
The name of your vCenter host (for example: `vc1.example.com`). You should already know the value for this setting.
`:vsphere_username`
Your vCenter username. You should already know the value for this setting.
`:vsphere_password`
Your vCenter password. You should already know the value for this setting.
`:vsphere_expected_pubkey_hash`
A public key hash for your vSphere server. The value for this setting can be obtained by filling the other three settings and running the following command:

```
$ puppet node_vmware list
```

This will result in an error message containing the server's public key hash…

```
notice: Connecting ...·
```

```
err: The remote system presented a public key with hash
431dd5d0412aab11b14178290d9fcc5acb041d37f90f36f888de0cebfffff0a8 but
we're expecting a hash of <unset>.  If you are sure the remote system is
authentic set vsphere_expected_pubkey_hash: <the hash printed in this
message> in ~/.fog
err: Try 'puppet help node_vmware list' for usage
```

…which can then be entered as the value of this setting.

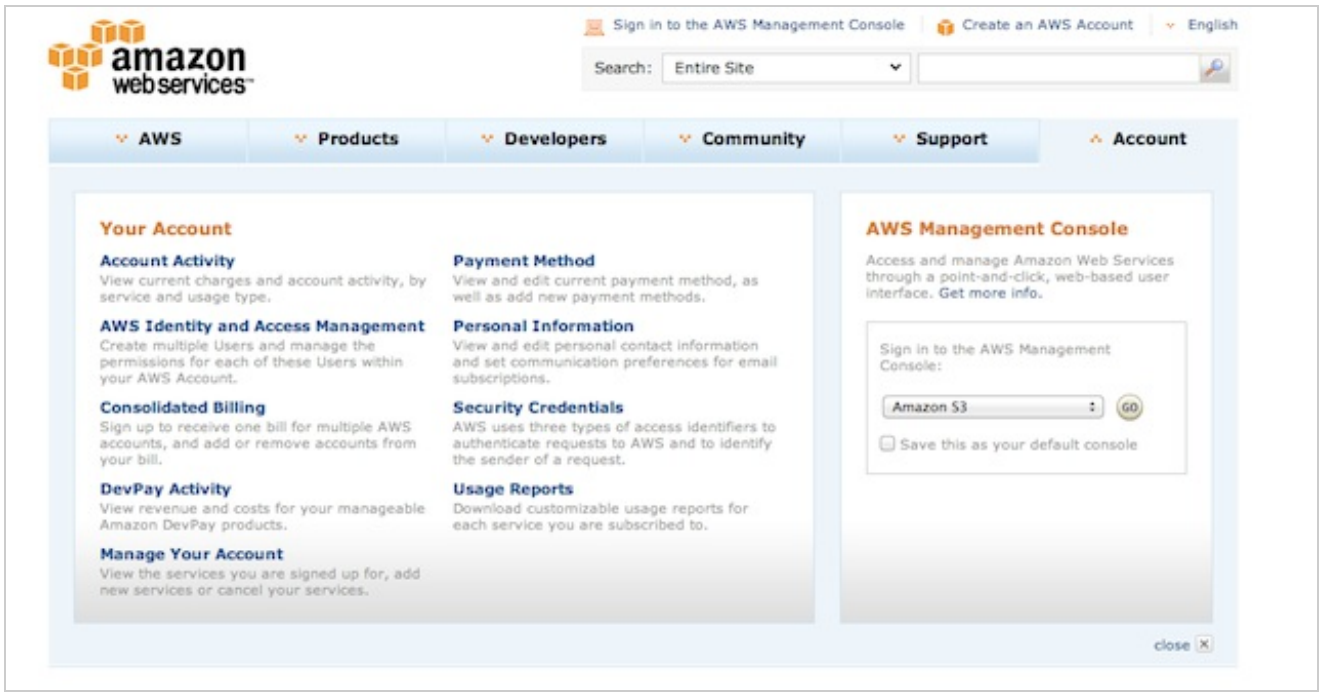
Adding Amazon Web Services credentials

To connect to Amazon Web Services, you must put the following information in your `~/.fog` file:

`:aws_access_key_id`
Your AWS Access Key ID. See below for how to find this.

`:aws_secret_access_key`
Your AWS Secret Key ID. See below for how to find this.

You can get find your Amazon Web Services credentials online in your Amazon account. To view them, go to Amazon AWS and click on the Account tab.



Select the Security Credentials menu and from there choose the “Access Credentials” section; click on the “Access Keys” tab to view your Access Keys.

You need to record two pieces of information: the Access Key ID and the Secret Key ID. To see your Secret Access Key, just click the “Show” link under “Secret Access Key”.

Put both keys in your `~/.fog` file as described above.

Additional AWS Configuration

To provision with Puppet, your Amazon Web Services EC2 account will need to have:

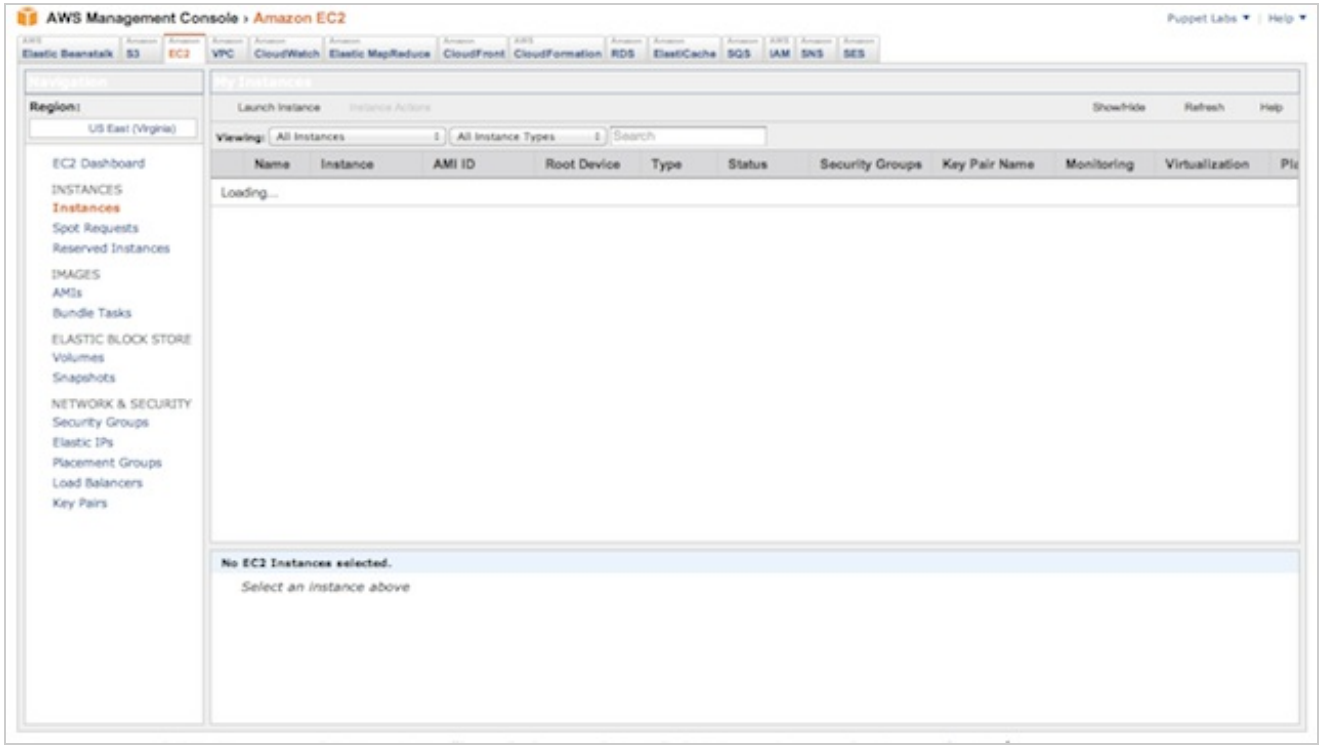
- At least one Amazon-managed SSH key pair.
- A security group that allows outbound traffic on ports 8140 and 61613, and inbound SSH traffic

(port 22) from the machine being used for provisioning.

You'll need to provide the names of these resources as arguments when running the provisioning commands.

KEY PAIRS

To find or create your Amazon SSH key pair, browse to the Amazon Web Service EC2 console.

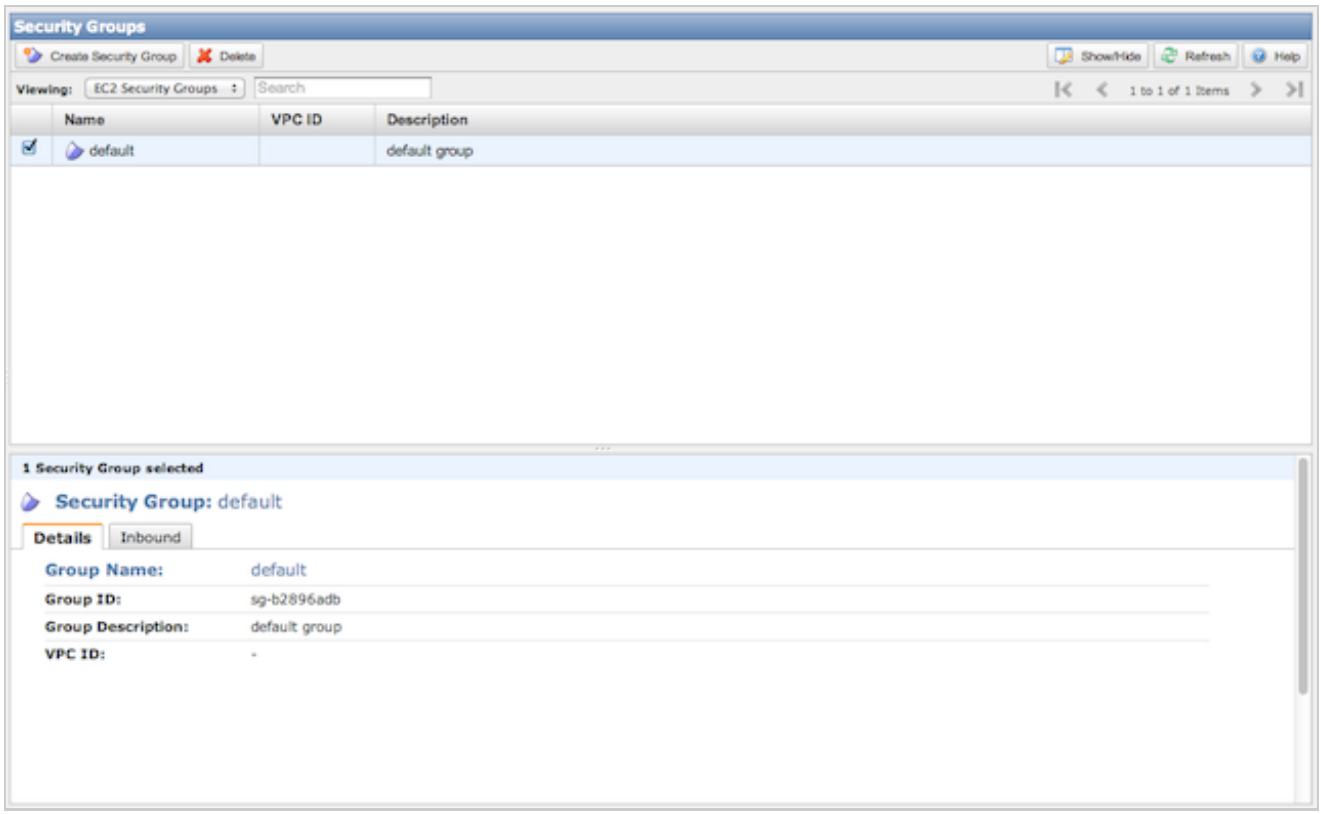


Select the "Key Pairs" menu item from the dashboard. If you don't have any existing key pairs, you can create one with the "Create Key Pairs" button. Specify a new name for the key pair to create it; the private key file will be automatically downloaded to your host.

Make a note of the name of your key pair, as it is used when creating new instances.

SECURITY GROUP

To add or edit a security group, select the "Security Groups" menu item from the dashboard. You should see a list of the available security groups. If no groups exist, you can create a new one by clicking the "Create Security Groups" button; otherwise, you can edit an existing group.



To add the required rules, select the “Inbound” tab and add an SSH rule. You can also specify a specific source to lock the source IP down to an appropriate source IP or network. Click “Add Rule” to add the rule, then click “Apply Rule Changes” to save.

You should also ensure that your security group allows outbound traffic on ports 8140 and 61613. These are the ports PE uses to request configurations and listen for orchestration messages.

Troubleshooting

Missing .fog file or credentials

If you attempt to provision without creating a `.fog` file or without populating the file with the appropriate credentials:

For VMware you’ll see the following error:

```
$ puppet node_vmware list
notice: Connecting ...
err: Missing required arguments: vsphere_username, vsphere_password,
vsphere_server
err: Try 'puppet help node_vmware list' for usage
```

For Amazon Web Services you’ll see the following error:

```
$ puppet node_aws list
err: Missing required arguments: aws_access_key_id,
aws_secret_access_key
```

```
err: Try 'puppet help node_aws list' for usage
```

Add the appropriate file or missing credentials to the file to resolve this issue.

← Cloud Provisioning: Overview — Index — Cloud Provisioning: Provisioning with VMware →

PE 2.0 » Cloud Provisioning » VMware Provisioning

← Cloud Provisioning: Configuring and Troubleshooting — Index — Cloud Provisioning: Provisioning with AWS →

Provisioning With VMware

Puppet Enterprise can create and manage VMware virtual machines on your vSphere server using vCenter.

If you're new to VMware vSphere then we recommend looking at the vSphere documentation.

Listing VMware vSphere Instances

Let's get started by listing the machines currently on our vSphere server. We do this by running the `puppet node_vmware list` command.

```
$ puppet node_vmware list
```

If you haven't yet confirmed your vSphere server's public key hash in your `~/.fog` file, you'll receive an error message containing said hash:

```
$ puppet node_vmware list
notice: Connecting ...·
err: The remote system presented a public key with hash
431dd5d0412aab11b14178290d9fcc5acb041d37f90f36f888de0cebfffff0a8 but
we're expecting a hash of <unset>.  If you are sure the remote system is
authentic set vsphere_expected_pubkey_hash: <the hash printed in this
message> in ~/.fog
err: Try 'puppet help node_vmware list' for usage
```

Confirm that you are communicating with a trusted vSphere server by checking the hostname in your `~/.fog` file, then add the hash to your `.fog` file as follows:

```
:vsphere_expected_pubkey_hash:
431dd5d0412aab11b14178290d9fcc5acb041d37f90f36f888de0cebfffff0a8
```

Now we can run the `puppet node_vmware list` command and see a list of our existing virtual machines:

```
$ puppet node_vmware list
notice: Connecting ...
notice: Connected to vc01.example.com as cloudprovisioner (API version 4.1)
notice: Finding all Virtual Machines ... (Started at 12:16:01 PM)
notice: Control will be returned to you in 10 minutes at 12:26 PM if locating
is unfinished.
Locating:          100% |ooooooooooooooooooooooooooooooooooooooooooooooooooo|
Time: 00:00:34
notice: Complete
/Datacenters/Solutions/vm/master_template
```

```
powerstate: poweredOff
name:       master_template
hostname:   puppetmaster.example.com
instanceid: 5032415e-f460-596b-c55d-6ca1d2799311
ipaddress:  ---.---.---.---
template:   true

/Datacenters/Solutions2/vm/puppetagent
powerstate: poweredOn
name:       puppetagent
hostname:   agent.example.com
instanceid: 5032da5d-68fd-a550-803b-aa6f52fbf854
ipaddress:  192.168.100.218
template:   false
```

We can see that we've connected to our vSphere server and returned a VMware template and a virtual machine. VMware templates contain the information needed to build new virtual machines, such as the operating system, hardware configuration, and other details. A virtual machine is an existing machine that has already been provisioned on the vSphere server.

The following information is returned:

- The location of the template or machine
- The status of the machine (for example, poweredOff or poweredOn)
- The name of the template or machine on the vSphere server
- The host name of the machine
- The instanceid of the machine
- The IP address of the machine (note that templates don't have IP addresses)
- The type of entry – either a VMware template or a virtual machine

Creating a New VMware Virtual Machine

Puppet Enterprise can create and manage virtual machines from VMware templates. This is done with the `node_vmware create` action.

```
$ puppet node_vmware create --name=newpuppetmaster --
template="/Datacenters/Solutions/vm/master_template"
notice: Connecting ...
notice: Connected to vc01.example.com as cloudprovisioner (API version 4.1)
notice: Locating VM at /Datacenters/Solutions/vm/master_template (Started at
12:38:58 PM)
notice: Control will be returned to you in 10 minutes at 12:48 PM if locating
(1/2) is unfinished.
Locating (1/2):    100%
|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo| Time: 00:00:16
notice: Starting the clone process (Started at 12:39:15 PM)
notice: Control will be returned to you in 10 minutes at 12:49 PM if starting
(2/2) is unfinished.
Starting (2/2):    100%
```

```
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooo| Time: 00:00:03
---
name: newpuppetmaster
power_state: poweredOff
...
status: success
```

Here we've created a new virtual machine named `newpuppetmaster` with a template of `/Datacenters/Solutions/vm/master_template`. (We saw this template earlier when we listed all the resources available on our vSphere server.) The virtual machine is now created and will be powered on. Powering on may take several minutes to complete.

Starting, Stopping and Terminating VMware Virtual Machines

You can start, stop, and terminate virtual machines with the `start`, `stop`, and `terminate` actions.

To start a virtual machine:

```
$ puppet node_vmware start /Datacenters/Solutions/vm/newpuppetmaster
```

You can see we've specified the path to the virtual machine we wish to start; in this case `/Datacenters/Solutions/vm/newpuppetmaster`.

To stop a virtual machine:

```
$ puppet node_vmware stop /Datacenters/Solutions/vm/newpuppetmaster
```

This will stop the running virtual machine (it may take a few minutes).

Lastly, we can terminate a VMware instance. Be aware this will:

- Force-shutdown the virtual machine
- Delete the virtual machine AND its hard disk images

This is a destructive action that should only be taken when you wish to delete the virtual machine!

Getting more help

The `puppet node_vmware` command has extensive in-line help documentation and a man page.

To see the available actions and command line options, run:

```
$ puppet help node_vmware
USAGE: puppet node_vmware <action>

This subcommand provides a command line interface to work with VMware vSphere
Virtual Machine instances.  The goal of these actions is to easily create
new virtual machines, install Puppet onto them, and clean up when they're
no longer required.

OPTIONS:
--mode MODE                    - The run mode to use (user, agent, or master).
--render-as FORMAT             - The rendering format to use.
--verbose                      - Whether to log verbosely.
--debug                        - Whether to log debug information.

ACTIONS:
create       Create a new VM from a template
find         Find a VMware Virtual Machine
list         List VMware Virtual Machines
start        Start a Virtual Machine
stop         Stop a running Virtual Machine
```

```
terminate    Terminate (destroy) a VM

See 'puppet man node_vmware' or 'man puppet-node_vmware' for full help.
```

You can also view the man page for more detailed help.

```
$ puppet man node_vmware
```

You can get help on individual actions by running:

```
$ puppet help node_vmware <ACTION>
```

For example:

```
$ puppet help node_vmware start
```

← Cloud Provisioning: Configuring and Troubleshooting — Index — Cloud Provisioning: Provisioning with AWS →

PE 2.0 » Cloud Provisioning » AWS Provisioning

← Cloud Provisioning: Provisioning with VMware — Index — Cloud Provisioning: Classifying Nodes and Remotely Installing PE →

Provisioning With Amazon Web Services

Puppet Enterprise can create and manage EC2 machine instances using Amazon Web Services.

If you are new to Amazon Web Services, we recommend reading the Getting Started documentation.

Listing Amazon EC2 instances

Let's start by listing our running EC2 instances. We do this by running the `puppet node_aws list` command.

```
$ puppet node_aws list
i-013eb462:
  created_at: Sat Nov 12 02:10:06 UTC 2011
  dns_name: ec2-107-22-110-102.compute-1.amazonaws.com
  id: i-013eb462
  state: running
i-019f0a62:
  created_at: Sat Nov 12 03:48:50 UTC 2011
  dns_name: ec2-50-16-145-167.compute-1.amazonaws.com
  id: i-019f0a62
  state: running
i-01a33662:
  created_at: Sat Nov 12 04:32:25 UTC 2011
  dns_name: ec2-107-22-79-148.compute-1.amazonaws.com
  id: i-01a33662
  state: running
```

Here we can see we've got three running EC2 instances and the following information has been returned:

- The instance name
- The date they were created on
- The DNS host name of the instance
- The ID of the instance
- The state of the instance, for example running or terminated

If you have no instances running, then nothing will be returned.

Creating a new Amazon EC2 instance

You can create a new instance with the `node_aws create` action.

To create a new EC2 instance we need to add three required options:

- The AMI image we wish to start.
- The name of the SSH key pair to start the image with (see here for more about creating Amazon-

managed key pairs).

- The type of instance we wish to create. You can see a list of types here.

Let's provide this information and run the command:

```
$ puppet node_aws create --image ami-edae6384 --keyname cloudprovisioner --type
m1.small
notice: Creating new instance ...
notice: Creating new instance ... Done
notice: Creating tags for instance ...
notice: Creating tags for instance ... Done
notice: Launching server i-df7ee898 ...
##################
notice: Server i-df7ee898 is now launched
notice: Server i-df7ee898 public dns name: ec2-50-18-93-82.us-east-
1.compute.amazonaws.com
ec2-50-18-93-82.us-east-1.compute.amazonaws.com
```

We've created a new instance using an AMI of `ami-edae6384`, a key named `cloudprovisioner` and of the type `m1.small`. If you've forgotten the available key names on your account, you can get a list with the `node_aws list_keynames` action:

```
$ puppet node_aws list_keynames
cloudprovisioner (ad:d4:04:9f:b0:8d:e5:4e:4c:46:00:bf:88:4f:b6:c2:a1:b4:af:56)
```

You can also specify a variety of other options, including the region to start the instance in, and you can see a full list of these options by running `puppet help node_aws create`.

After the instance has been created, the public DNS name of the instance will be returned. In this case: `ec2-50-18-93-82.us-east-1.compute.amazonaws.com`.

Connecting to an EC2 instance

Once you've created an EC2 instance you can then connect to it using SSH. To do this we need the private key we downloaded earlier from the Amazon Web Services console. Add this key to your local SSH configuration, usually in the `.ssh` directory.

```
$ cp mykey.pem ~/.ssh/mykey.pem
```

Ensure the `.ssh` directory and the key have appropriate permissions.

```
$ chmod 0700 ~/.ssh
$ chmod 0600 ~/.ssh/mykey.pem
```

We can now use this key to connect to our new instance.

```
$ ssh -i ~/.ssh/mykey.pem root@ec2-50-18-93-82.us-east-1.compute.amazonaws.com
```

Terminating an EC2 instance

Once you've finished with an EC2 instance you can easily terminate it. Terminating an instance destroys the instance entirely and is a destructive action that should only be performed when you've finished with the instance. To terminate an instance, use the `node_aws terminate` action.

```
$ puppet node_aws terminate ec2-50-18-93-82.us-east-1.compute.amazonaws.com
notice: Destroying i-df7ee898 (ec2-50-18-93-82.us-east-1.compute.amazonaws.com)
...
notice: Destroying i-df7ee898 (ec2-50-18-93-82.us-east-1.compute.amazonaws.com)
... Done
```

Getting more help

The `puppet node_aws` command has extensive in-line help documentation and a man page.

To see the available actions and command line options, run:

```
$ puppet help node_aws
USAGE: puppet node_aws <action>

This subcommand provides a command line interface to work with Amazon EC2
machine instances.  The goal of these actions are to easily create new
machines, install Puppet onto them, and tear them down when they're no longer
required.

OPTIONS:
  --mode MODE                    - The run mode to use (user, agent, or
master).
  --render-as FORMAT             - The rendering format to use.
  --verbose                      - Whether to log verbosely.
  --debug                        - Whether to log debug information.

ACTIONS:
  bootstrap        Create and initialize an EC2 instance using Puppet
  create           Create a new EC2 machine instance.
  fingerprint      Make a best effort to securely obtain the SSH host key
                   fingerprint
  list             List AWS EC2 node instances
  list_keynames    List available AWS EC2 key names
  terminate        Terminate an EC2 machine instance

See 'puppet man node_aws' or 'man puppet-node_aws' for full help.
```

You can also view the man page for more detailed help.

```
$ puppet man node_aws
```

You can get help on individual actions by running:

```
$ puppet help node_aws <ACTION>
```

For example,

```
$ puppet help node_aws list
```

← Cloud Provisioning: Provisioning with VMware — Index — Cloud Provisioning: Classifying Nodes and Remotely Installing PE →

PE 2.0 » Cloud Provisioning » Classifying and Installing

← Cloud Provisioning: Provisioning with AWS — Index — Cloud Provisioning: Man Page: puppet node_vmware →

Classifying New Nodes and Remotely Installing PE

In addition to the provisioning actions described in the prior chapters, Puppet Enterprise includes actions for adding nodes to a pre-existing console group and remotely installing Puppet Enterprise on new nodes.

Classifying nodes

Once you have created virtual machines or instances, you can add them to a group in the console right from the command line. This has the same effect as adding nodes to a group with the console's web interface, but can be more convenient if you're already at the command line and have the node's name ready at hand.

To classify nodes and add them to a console group, use the `puppet node classify` command.

```
$ puppet node classify \
--node-group=appserver_pool \
--enc-server=localhost \
--enc-port=443 \
--enc-ssl \
--enc-auth-user=console \
--enc-auth-passwd=password \
ec2-50-19-149-87.compute-1.amazonaws.com

notice: Contacting https://localhost:443/ to classify
ec2-50-19-149-87.compute-1.amazonaws.com
complete
```

Here we're adding an AWS EC2 instance to the console.

With the command's options, we specify:

- The `--node-group` we'd like to add the node to (in our case the `appserver_pool` group)
- The console's server
- The console's port
- Whether the console uses SSL
- The user name for connecting to the console
- The password for connecting to the console — this user name and password were set when the console was installed

As the command's argument, we specify the name of the node we're classifying. If we now navigate to the console's web interface, we can see this host has been added to the `appserver_pool` group.

To see additional help for node classification, run `puppet help node classify`. For more about how the console groups and classifies nodes, see the chapter of this manual about grouping and classifying.

Installing Puppet

Use the `puppet node install` command to install Puppet Enterprise (or open-source Puppet) onto new nodes.

```
$ puppet node install --keyfile=~/.ssh/mykey.pem --login=root ec2-50-19-207-
181.compute-1.amazonaws.com
notice: Waiting for SSH response ...
notice: Waiting for SSH response ... Done
notice: Installing Puppet ...
puppetagent_certname: ec2-50-19-207-181.compute-1.amazonaws.com-ee049648-3647-
0f93-782b-9f30e387f644
status: success
```

With the command's options, we specify:

- An SSH key to log in with (`--keyfile`) — the `install` action uses SSH to connect to the host, and needs an SSH key. For VMware, this key should be loaded onto the template you used to create your virtual machine. For Amazon EC2, it should be the private key from the key pair you used to create the instance.
- The local user account to log in as (`--login`).

As the command's argument, we specify the name of the node we're installing Puppet Enterprise on.

For the default installation, the `install` action uses packages provided by Puppet Labs and stored

in Amazon S3 storage. You can also specify packages located on your local host, or on a share in your local network. See `puppet help node install` or `puppet man node` for more details.

In addition to these default configuration options, we can specify a number of additional options to control how and what we install on the host. We can specify the specific version of Puppet Enterprise (or we can install open source Puppet too) to install. We can also control the version of Facter to install, the specific answers file to use to configure Puppet Enterprise, the certificate name of the agent to be installed and a variety of other options. To see a full list of the available options, use the `puppet help node install` command.

Classifying and Installing Puppet in One Command

Rather than using multiple commands to classify and install Puppet on a node, you can use the `init` action, which performs both actions in a single command. For example:

```
$ puppet node init \
--node-group=appserver_pool \
--enc-server=localhost \
--enc-port=443 \
--enc-ssl \
--enc-auth-user=console \
--enc-auth-passwd=password \
--keyfile=~/.ssh/mykey.pem \
--login=root \
ec2-50-19-207-181.compute-1.amazonaws.com
```

The `init` action combines the options for the `classify` and `install` actions. The invocation above will connect to the console, classify the node in the `appserver_pool` group, and then install Puppet Enterprise on this node.

PE 2.0 » Cloud Provisioning » Man » node_vmware

Man Page: node_vmware

NAME

`puppet-node_vmware` – View and manage VMware vSphere virtual machine nodes.

SYNOPSIS

puppet node_vmware action

DESCRIPTION

This subcommand provides a command line interface to work with VMware vSphere virtual machine instances. The goal of these actions is to easily create new virtual machines, start and stop them, and clean up when they're no longer required. After creating new nodes, you can use the `node` subcommand's `install`, `classify`, and `init` actions to bring them into your Puppet infrastructure.

To use this subcommand, you must include four configuration options in your ~/.fog file: A vCenter user account, a password, a server hostname, and the hash digest of the vCenter server's SSL public key. (See below for the exact names of the settings.) Puppet will display the public key digest the first time you connect to vCenter if the setting is omitted from the configuration file.

The following is an example of ~/.fog configured with an "API" access account.

```
:default:
  :vsphere_server: vcenter.example.lan
  :vsphere_username: api_user
  :vsphere_password: sekret
  :vsphere_expected_pubkey_hash: 431dd5d0412...
```

OPTIONS

Note that any configuration parameter that's valid in the configuration file is also a valid long argument, although it may or may not be relevant to the present action. For example, `server` is a valid configuration parameter, so you can specify `--server <servername>` as an argument.

See the configuration file documentation at http://docs.puppetlabs.com/references/stable/configuration.html for the full list of acceptable parameters. A commented list of all configuration options can also be generated by running puppet with `--genconfig`.

--mode MODE
The run mode to use for the current action. Valid modes are `user`, `agent`, and `master`.
--render-as FORMAT
The format in which to render output. The most common formats are `json`, `s` (string), `yaml`, and

`console`, but other options such as `dot` are sometimes available.
--verbose
Whether to log verbosely.
--debug
Whether to log debug information.

ACTIONS

`create` - Create a new VM from a template
`SYNOPSIS`

puppet node_vmware create --name= | -n= --template= | -t= [--wait | -w] [--wait-for-boot | -i] [--timeout=]

`DESCRIPTION`

This action creates a new virtual machine by cloning an existing VMware VM marked as a template.

`OPTIONS`

--name= | -n= - The name of the new VM to create. This name must be unique within the folder containing the source VM or template.

--template= | -t= - The path of the VMware virtual machine template to clone. A full path to the template is expected in the same format as displayed from the list action.

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.

--wait | -w - If this option is specified then Puppet will wait for the VMware copy operation to finish. This could be a lengthy process but provides the ability to reliably wait until the VM has finished cloning before moving on.

--wait-for-boot | -i - If this option is specified then Puppet will wait for the VMware copy to finish and the VM to boot and obtain an IP address on the network. This requires the VMware Tools installed in the guest VM. This option implies the --wait option.

`RETURNS`

Hash containing attributes about the new virtual machine.
`find` - Find a VMware virtual machine
`SYNOPSIS`

puppet node_vmware find [--timeout=] path

`DESCRIPTION`

Find a single VMware virtual machine by path. The default output of the find action provides a limited set of information. To obtain all available attributes for a virtual machine, please use a

different rendering mode such as --render-as yaml.□

`OPTIONS`

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.□

`RETURNS`

Array of virtual machine attribute hashes.
`list` - List VMware virtual machines
`SYNOPSIS`

puppet node_vmware list [--folder= | -f=] [--timeout=]

`DESCRIPTION`

List VMware virtual machines in all Datacenters. This command may take considerable time to run in large VMware environments.

The default output of the list action provides a limited set of information. To obtain all available attributes for each virtual machine, please use a different rendering mode, such as --render-as□ yaml.

`OPTIONS`

--folder= | -f= - This option limits the listing of virtual machines to a single folder, which is much faster than listing all virtual machines in the entire inventory.

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.□

`RETURNS`

Array of virtual machine attribute hashes.
`start` - Start a virtual machine
`SYNOPSIS`

puppet node_vmware start [--timeout=] path

`DESCRIPTION`

This action starts a stopped virtual machine. This is also known as powering on the virtual machine.

`OPTIONS`

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.□

`RETURNS`

Hash containing the VM's status
`stop` - Stop a running virtual machine
`SYNOPSIS`

puppet node_vmware stop [--force] [--timeout=] path

`DESCRIPTION`

This action stops a running virtual machine. By default, the guest operating system will be gracefully shut down using VMware Tools.

`OPTIONS`

--force - Forcibly power off the virtual machine without giving the guest operating system a chance to gracefully shut down.

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.

`RETURNS`

Hash containing the VM's status
`terminate` - Terminate (destroy) a VM
`SYNOPSIS`

puppet node_vmware terminate [--timeout=] path

`DESCRIPTION`

This action terminates a virtual machine. This is a destructive action that will forcibly shut down a VM if it's running and then delete it from the VMware inventory and datastore. Only carry out this action on VMs you want to permanently delete. The VM hard disk images WILL BE DELETED.

`OPTIONS`

--timeout= - The maximum number of seconds to wait for API calls to complete. Many of the API calls may take considerable time to run. If not specified, this defaults to 600 seconds.

`RETURNS`

Hash containing the VM's status

EXAMPLES

`create`

$ puppet node_vmware create --name jefftest --template /Datacenters/Solutions/vm/el56

`find`

Get info about a specific VM:□

```
$ puppet node_vmware find /Datacenters/Solutions/vm/el56
/Datacenters/Solutions/vm/el56
  powerstate: poweredOff
  name:       el56
  hostname:   --------
  instanceid: 50323f93-6835-1178-8b8f-9e2109890e1a
  ipaddress:  ---.---.---.---
  template:   true
```

`list`

List all VMs in the entire inventory:

```
$ puppet node_vmware list
/Datacenters/Solutions/vm/centos5-01
  powerstate: poweredOff
  name:       centos5-01
  hostname:   centos5.example.vm
  instanceid: 503271e5-2cc4-1f1a-4303-75d33938fcf4
  ipaddress:  ---.---.---.---
  template:   false

/Datacenters/Solutions/vm/el56
  powerstate: poweredOff
  name:       el56
  hostname:   --------
  instanceid: 50323f93-6835-1178-8b8f-9e2109890e1a
  ipaddress:  ---.---.---.---
  template:   true
```

List only the VMs in the Templates folder:

```
$ puppet node_vmware list --folder=/Datacenters/DC1/vm/Templates
/Datacenters/Solutions/vm/Templates/el56template
  powerstate: poweredOff
  name:       el56template
  hostname:   --------
  instanceid: 09CB9D9B-2498-41E0-807A-382408829C07
  ipaddress:  ---.---.---.---
  template:   true
```

`start`

$ puppet node_vmware start /Datacenters/MyDC/vm/jefftest□

`stop`

$ puppet node_vmware stop /Datacenters/MyDC/vm/jefftest --force□

`terminate`

$ puppet node_vmware terminate /Datacenters/MyDC/vm/jefftest□

COPYRIGHT AND LICENSE

← Cloud Provisioning: Classifying Nodes and Remotely Installing PE — Index — Cloud Provisioning: Man Page: puppet node_aws →

PE 2.0 » Cloud Provisioning » Man » node_aws

← Cloud Provisioning: Man Page: puppet node_vmware — Index — Cloud Provisioning: Man Page: puppet node →

Man Page: node_aws

NAME

`puppet-node_aws` – View and manage Amazon AWS EC2 nodes.

SYNOPSIS

puppet node_aws action

DESCRIPTION

This subcommand provides a command line interface to work with Amazon EC2 machine instances. The goal of these actions is to easily create new machines, install Puppet onto them, and tear them down when they're no longer required.

OPTIONS

Note that any configuration parameter that's valid in the configuration file is also a valid long argument, although it may or may not be relevant to the present action. For example, `server` is a valid configuration parameter, so you can specify `--server <servername>` as an argument.

See the configuration file documentation at http://docs.puppetlabs.com/references/stable/configuration.html for the full list of acceptable parameters. A commented list of all configuration options can also be generated by running puppet with `--genconfig`.

--mode MODE
The run mode to use for the current action. Valid modes are `user`, `agent`, and `master`.
--render-as FORMAT
The format in which to render output. The most common formats are `json`, `s` (string), `yaml`, and `console`, but other options such as `dot` are sometimes available.
--verbose
Whether to log verbosely.
--debug
Whether to log debug information.

ACTIONS

`bootstrap` – Create and initialize an EC2 instance using Puppet.
`SYNOPSIS`

puppet node_aws bootstrap [--platform=] [--region=] --image= | -i= --type= --keyname= [--group= | -g= | --security-group=] --login= | -l= | --username= --keyfile= [--installer-payload=] [--installer-answers=] [--puppetagent-certname=] [--install-script=] [--puppet-version=] [--pe-version=] [--facter-version=] [--enc-ssl] [--enc-server=] [--enc-port=] [--enc-auth-user=] [--enc-auth-passwd=] [--node-group= | --as=]

DESCRIPTION

Creates an instance, classifies it, and signs its certificate. The classification is currently done using Puppet Dashboard or Puppet Enterprise's console.

OPTIONS

--enc-auth-passwd= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

Note: This option will default to the PUPPET_ENC_AUTH_PASSWD environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

--enc-auth-user= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

Note: This option will default to the PUPPET_ENC_AUTH_USER environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

--enc-port= - The port of the External Node Classifier. This currently only supports Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

--enc-server= - The hostname of the external node classifier. This currently only supports Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

--enc-ssl - By default, we do not connect to the ENC over SSL. This option configures all HTTP connections to the ENC to use SSL in order to provide encryption. This option should be set when using Puppet Enterprise 2.0 and higher.

--facter-version= - The version of facter that should be installed. This only makes sense in open source installation mode.

--group= | -g= | --security-group= - The security group(s) that the machine will be associated with. A security group determines the rules for both inbound and outbound connections.

Multiple groups can be specified as a colon-separated list.

--image= | -i= - The pre-configured operating system image to use when creating this machine instance. Currently, only AMI images are supported. Example of a Redhat 5.6 32bit image: ami-b241bfdb

--install-script= - Name of the installation template to use when installing Puppet. The current list of supported templates is: gems, puppet-enterprise

--installer-answers= - Location of the answers file that should be copied to the machine to install Puppet Enterprise.

--installer-payload= - Location of the Puppet enterprise universal tarball to be used for the installation. Can be a local file path or a URL. This option is only required if Puppet should be installed on the machine. The tarball specified must be gzipped.

--keyfile= - The filesystem path to a local private key that can be used to SSH into the node. If the node was created with the `node_aws create` action, this should be the path to the private key file downloaded from the Amazon EC2 interface.

Specify 'agent' if you have the key loaded in ssh-agent and available via the SSH_AUTH_SOCK variable.

--keyname= - The name of the SSH key pair to use, as listed in the Amazon AWS console. When creating the instance, Amazon will install the requested SSH public key into the instance's authorized_keys file. Not to be confused with the --keyfile option of the `node` subcommand's `install` action.

You can use the `list_keynames` action to get a list of valid key pairs.

--login= | -l= | --username= - The name of the user Puppet should use when logging in to the node. This user should configured to allow passwordless access via the SSH key supplied in the -i --keyfile option.

This is usually the root user.

--node-group= | --as= - The PE console or Puppet Dashboard group to associate the node with. The group must already exist in the ENC, or an error will be returned. If the node has not been registered with the ENC, it will automatically be registered when assigning it to a group.

--pe-version= - Version of Puppet Enterprise to be passed to the installer script. Defaults to 1.1.

--platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--puppet-version= - Version of Puppet to be installed. This version is passed to the Puppet installer script.

--puppetagent-certname= - This option allows you to specify an optional puppet agent certificate name to configure on the target system. This option applies to the puppet-enterprise and puppet-enterprise-http installation scripts. If provided, this option will replace any puppet agent certificate name provided in the puppet enterprise answers file. This certificate name will show up in the console (or Puppet Dashboard) when the agent checks in for the first time.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland),

Asia Pacific (Singapore), and Asia Pacific (Tokyo).□

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security□ groups to allow SSH access.

--type= - Type of instance to be launched. The type specifies characteristics that a machine will□ have, such as architecture, memory, processing power, storage, and IO performance. The type selected will determine the cost of a machine instance. Supported types are: 'm1.small','m1.large','m1.xlarge','t1.micro','m2.xlarge', 'm2.2xlarge','x2.4xlarge','c1.medium','c1.xlarge','cc1.4xlarge'.
`create` - Create a new EC2 machine instance.
`SYNOPSIS`

puppet node_aws create [--platform=] [--region=] --image= | -i= --type= --keyname= [--group= | -g= | --security-group=]

`DESCRIPTION`

This action launches a new Amazon EC2 instance and returns the public DNS name suitable for SSH access.

A newly created system may not be immediately ready after launch while it boots. You can use the `fingerprint` action to wait for the system to become ready after launch.

If creation of the instance fails, Puppet will automatically clean up after itself and tear down the instance.

`OPTIONS`

--group= | -g= | --security-group= - The security group(s) that the machine will be associated with. A security group determines the rules for both inbound and outbound connections.

Multiple groups can be specified as a colon-separated list.□

--image= | -i= - The pre-configured operating system image to use when creating this machine instance. Currently, only AMI images are supported. Example of a Redhat 5.6 32bit image: ami-b241bfdb

--keyname= - The name of the SSH key pair to use, as listed in the Amazon AWS console. When creating the instance, Amazon will install the requested SSH public key into the instance's authorized_keys file. Not to be confused with the --keyfile option of the `node` subcommand's `install` action.

You can use the `list_keynames` action to get a list of valid key pairs.

--platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security groups to allow SSH access.

--type= - Type of instance to be launched. The type specifies characteristics that a machine will have, such as architecture, memory, processing power, storage, and IO performance. The type selected will determine the cost of a machine instance. Supported types are: 'm1.small','m1.large','m1.xlarge','t1.micro','m2.xlarge', 'm2.2xlarge','x2.4xlarge','c1.medium','c1.xlarge','cc1.4xlarge'.
`fingerprint` - Make a best effort to securely obtain the SSH host key fingerprint.
`SYNOPSIS`

puppet node_aws fingerprint [--platform=] [--region=] instance_name

`DESCRIPTION`

This action attempts to retrieve a host key fingerprint by using the EC2 API to search the console output. This provides a secure way to retrieve the fingerprint from an EC2 instance. You should run the `fingerprint` action immediately after creating an instance, as you wait for it to finish booting.

This action can only retrieve a fingerprint if the instance's original image was configured to print the fingerprint to the system console. Note that many machine images do not print the fingerprint to the console. If this action is unable to find a fingerprint, it will display a warning.

In either case, if this command returns without an error, then the instance being checked is ready for use.

`OPTIONS`

--platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-

southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security groups to allow SSH access.

`list` - List AWS EC2 machine instances.

`SYNOPSIS`

puppet node_aws list [--platform=] [--region=]

`DESCRIPTION`

This action obtains a list of instances from the cloud provider and displays them on the console output. For EC2 instances, only the instances in a specific region are provided.

`OPTIONS`

--platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security groups to allow SSH access.

`RETURNS`

Array of attribute hashes containing information about each EC2 instance.

`list_keynames` - List available AWS EC2 key names.

`SYNOPSIS`

puppet node_aws list_keynames [--platform=] [--region=]

`DESCRIPTION`

This action lists the available AWS EC2 key names and their fingerprints. Any key name from this list is a valid argument for the create action's --keyname option.

`OPTIONS`

--platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland),

Asia Pacific (Singapore), and Asia Pacific (Tokyo).

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security groups to allow SSH access.

`RETURNS`

Array of attribute hashes containing information about each key pair
`terminate` - Terminate an EC2 machine instance.
`SYNOPSIS`

puppet node_aws terminate [--region=] [--platform=] [--force | -f] instance_name

`DESCRIPTION`

Terminate the instance identified by instance_name.

`OPTIONS`

--force | -f - Forces termination of an instance. --platform= - The Cloud platform used to create new machine instances. Currently, AWS (Amazon Web Services) is the only supported platform.

--region= - The instance may run in any region EC2 operates within. The regions at the time of this documentation are: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).

The region names for this command are: eu-west-1, us-east-1, ap-northeast-1, us-west-1, ap-southeast-1

Note: to use another region, you will need to copy your keypair and reconfigure the security groups to allow SSH access.

EXAMPLES

`list`

List every instance in the US East region:

```
$ puppet node_aws list --region=us-east-1
i-e8e04588:
  created_at: Tue Sep 13 01:21:16 UTC 2011
  dns_name: ec2-184-72-85-208.compute-1.amazonaws.com
  id: i-e8e04588
  state: running
```

`list_keynames`

List the available key pairs:

```
$ puppet node_aws list_keynames
cody (58:c6:4f:3e:b5:51:e0:ec:49:55:4e:98:43:8f:28:f3:9a:14:c8:a3)
jeff (6e:b6:0a:27:5b:67:cd:8b:47:74:9c:f7:b2:b0:b9:ab:3a:25:d0:28)
matt (4b:8c:8d:a9:e5:88:6a:47:b7:8b:97:c5:77:e7:b7:6f:fd:b9:64:b3)
```

Get the key pair list as an array of JSON hashes:

```
$ puppet node_aws list_keynames --render-as json
[{"name":"cody","fingerprint":"58:c6:4f:3e:b5:51:e0:ec:49:55:4e:98:43:8f:28:f3:9a:


{"name":"jeff","fingerprint":"6e:b6:0a:27:5b:67:cd:8b:47:74:9c:f7:b2:b0:b9:ab:3a:2


{"name":"matt","fingerprint":"4b:8c:8d:a9:e5:88:6a:47:b7:8b:97:c5:77:e7:b7:6f:fd:b
```

COPYRIGHT AND LICENSE

← Cloud Provisioning: Man Page: puppet node_vmware — Index — Cloud Provisioning: Man Page: puppet node →

PE 2.0 » Cloud Provisioning » Man » node

← Cloud Provisioning: Man Page: puppet node_aws — Index — Compliance: Basics and UI →

Man Page: node

NAME

`puppet-node` - View and manage node definitions.

SYNOPSIS

puppet node action [--terminus TERMINUS] [--extra HASH]

DESCRIPTION

This subcommand interacts with node objects, which are used by Puppet to build a catalog. A node object consists of the node's facts, environment, node parameters (exposed in the parser as top-scope variables), and classes.

OPTIONS

Note that any configuration parameter that's valid in the configuration file is also a valid long argument, although it may or may not be relevant to the present action. For example, `server` is a valid configuration parameter, so you can specify `--server <servername>` as an argument.

See the configuration file documentation at http://docs.puppetlabs.com/references/stable/configuration.html for the full list of acceptable parameters. A commented list of all configuration options can also be generated by running puppet with `--genconfig`.

--mode MODE
The run mode to use for the current action. Valid modes are `user`, `agent`, and `master`.
--render-as FORMAT
The format in which to render output. The most common formats are `json`, `s` (string), `yaml`, and `console`, but other options such as `dot` are sometimes available.
--verbose
Whether to log verbosely.
--debug
Whether to log debug information.
--extra HASH
A terminus can take additional arguments to refine the operation, which are passed as an arbitrary hash to the back-end. Anything passed as the extra value is just send direct to the back-end.
--terminus TERMINUS
Indirector faces expose indirected subsystems of Puppet. These subsystems are each able to retrieve and alter a specific type of data (with the familiar actions of `find`, `search`, `save`, and `destroy`) from an arbitrary number of pluggable backends. In Puppet parlance, these backends are called terminuses.

Almost all indirected subsystems have a `rest` terminus that interacts with the puppet master's

data. Most of them have additional terminuses for various local data models, which are in turn used by the indirected subsystem on the puppet master whenever it receives a remote request.

The terminus for an action is often determined by context, but occasionally needs to be set explicitly. See the "Notes" section of this face's manpage for more details.

ACTIONS

- `classify` - Add a node to a console or Dashboard group.: `SYNOPSIS`

  puppet node classify [--terminus TERMINUS] [--extra HASH] [--enc-ssl] [--enc-server=] [--enc-port=] [--enc-auth-user=] [--enc-auth-passwd=] [--node-group= | --as=] certname

  `DESCRIPTION`

  Add node certname to a group in Puppet Dashboard, Puppet Enterprise's console, or any external node classifier that provides a similar API.

  Classification of a node will allow it to receive proper configurations on its next Puppet run. This action assumes that you have already created a console or Dashboard group with the classes the node should receive in its configuration catalog.

  This action can be used on both physical and virtual machines, and can be run multiple times for a single node. This action can be safely run before the `install` action.

  `OPTIONS`

  --enc-auth-passwd= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

  Note: This option will default to the PUPPET_ENC_AUTH_PASSWD environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

  --enc-auth-user= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

  Note: This option will default to the PUPPET_ENC_AUTH_USER environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

  --enc-port= - The port of the External Node Classifier. This currently only supports Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

  --enc-server= - The hostname of the external node classifier. This currently only supports

Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

--enc-ssl - By default, we do not connect to the ENC over SSL. This option configures all HTTP connections to the ENC to use SSL in order to provide encryption. This option should be set when using Puppet Enterprise 2.0 and higher.

--node-group= | --as= - The PE console or Puppet Dashboard group to associate the node with. The group must already exist in the ENC, or an error will be returned. If the node has not been registered with the ENC, it will automatically be registered when assigning it to a group.

- `clean` - Clean up everything a puppetmaster knows about a node: `SYNOPSIS`

  puppet node clean [--terminus TERMINUS] [--extra HASH] [--[no-]unexport] host1 [host2 ...]

  `DESCRIPTION`

  This includes

  - Signed certificates ($vardir/ssl/ca/signed/node.domain.pem)
  - Cached facts ($vardir/yaml/facts/node.domain.yaml)
  - Cached node stuff ($vardir/yaml/node/node.domain.yaml)
  - Reports ($vardir/reports/node.domain)
  - Stored configs: it can either remove all data from an host in your storedconfig database, or with --unexport turn every exported resource supporting ensure to absent so that any other host checking out their config can remove those exported configurations.

  This will unexport exported resources of a host, so that consumers of these resources can remove the exported resources and we will safely remove the node from our infrastructure.

  `OPTIONS`

  --[no-]unexport - Unexport exported resources

- `destroy` - Invalid for this subcommand.: `SYNOPSIS`

  puppet node destroy [--terminus TERMINUS] [--extra HASH] key

  `DESCRIPTION`

  Invalid for this subcommand.

- `find` - Retrieve a node object.: `SYNOPSIS`

  puppet node find [--terminus TERMINUS] [--extra HASH] host

  `DESCRIPTION`

  Retrieve a node object.

  `RETURNS`

A hash containing the node's `classes`, `environment`, `expiration`, `name`, `parameters` (its facts, combined with any ENC-set parameters), and `time`. When used from the Ruby API: a Puppet::Node object.

RENDERING ISSUES: Rendering as string and json are currently broken; node objects can only be rendered as yaml.

- `info` - Print the default terminus class for this face.: `SYNOPSIS`

  puppet node info [--terminus TERMINUS] [--extra HASH]

  `DESCRIPTION`

  Prints the default terminus class for this subcommand. Note that different run modes may have different default termini; when in doubt, specify the run mode with the '--mode' option.

- `init` - Install Puppet on a node and clasify it.: `SYNOPSIS`

  puppet node init [--terminus TERMINUS] [--extra HASH] --login= | -l= | --username= --keyfile= [--installer-payload=] [--installer-answers=] [--puppetagent-certname=] [--install-script=] [--puppet-version=] [--pe-version=] [--facter-version=] [--enc-ssl] [--enc-server=] [--enc-port=] [--enc-auth-user=] [--enc-auth-passwd=] [--node-group= | --as=]

  `DESCRIPTION`

  Installs Puppet on an arbitrary node (see "install"), classify it in Puppet Dashboard or Puppet Enterprise's console (see "classify"), and automatically sign its certificate request (using the `certificate` face's `sign` action).

  `OPTIONS`

  --enc-auth-passwd= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

  Note: This option will default to the PUPPET_ENC_AUTH_PASSWD environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

  --enc-auth-user= - PE's console and Puppet Dashboard can be secured using HTTP authentication. If the console or dashboard is configured with HTTP authentication, use this option to supply credentials for accessing it.

  Note: This option will default to the PUPPET_ENC_AUTH_USER environment variable. Please use this environment variable if you are concerned about usernames and passwords being exposed via the Unix process table.

--enc-port= - The port of the External Node Classifier. This currently only supports Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

--enc-server= - The hostname of the external node classifier. This currently only supports Puppet Enterprise's console and Puppet Dashboard as external node classifiers.

--enc-ssl - By default, we do not connect to the ENC over SSL. This option configures all HTTP connections to the ENC to use SSL in order to provide encryption. This option should be set when using Puppet Enterprise 2.0 and higher.

--facter-version= - The version of facter that should be installed. This only makes sense in open source installation mode.

--install-script= - Name of the installation template to use when installing Puppet. The current list of supported templates is: gems, puppet-enterprise

--installer-answers= - Location of the answers file that should be copied to the machine to install Puppet Enterprise.

--installer-payload= - Location of the Puppet enterprise universal tarball to be used for the installation. Can be a local file path or a URL. This option is only required if Puppet should be installed on the machine. The tarball specified must be gzipped.

--keyfile= - The filesystem path to a local private key that can be used to SSH into the node. If the node was created with the `node_aws create` action, this should be the path to the private key file downloaded from the Amazon EC2 interface.

Specify 'agent' if you have the key loaded in ssh-agent and available via the SSH_AUTH_SOCK variable.

--login= | -l= | --username= - The name of the user Puppet should use when logging in to the node. This user should configured to allow passwordless access via the SSH key supplied in the -i --keyfile option.

This is usually the root user.

--node-group= | --as= - The PE console or Puppet Dashboard group to associate the node with. The group must already exist in the ENC, or an error will be returned. If the node has not been registered with the ENC, it will automatically be registered when assigning it to a group.

--pe-version= - Version of Puppet Enterprise to be passed to the installer script. Defaults to 1.1.

--puppet-version= - Version of Puppet to be installed. This version is passed to the Puppet installer script.

--puppetagent-certname= - This option allows you to specify an optional puppet agent certificate name to configure on the target system. This option applies to the puppet-enterprise and puppet-enterprise-http installation scripts. If provided, this option will replace any puppet

agent certificate name provided in the puppet enterprise answers file. This certificate name will show up in the console (or Puppet Dashboard) when the agent checks in for the first time.

- `install` - Install Puppet on a running node.: `SYNOPSIS`

  puppet node install [--terminus TERMINUS] [--extra HASH] --login= | -l= | --username= --keyfile= [--installer-payload=] [--installer-answers=] [--puppetagent-certname=] [--install-script=] [--puppet-version=] [--pe-version=] [--facter-version=] hostname_or_ip

  `DESCRIPTION`

  Installs Puppet on an existing node at hostname_or_ip. It uses scp to copy installation requirements to the machine, and ssh to run the installation commmands remotely.

  This action can be used on both physical and virtual machines.

  `OPTIONS`

  --facter-version= - The version of facter that should be installed. This only makes sense in open source installation mode.

  --install-script= - Name of the installation template to use when installing Puppet. The current list of supported templates is: gems, puppet-enterprise

  --installer-answers= - Location of the answers file that should be copied to the machine to install Puppet Enterprise.

  --installer-payload= - Location of the Puppet enterprise universal tarball to be used for the installation. Can be a local file path or a URL. This option is only required if Puppet should be installed on the machine. The tarball specified must be gzipped.

  --keyfile= The filesystem path to a local private key that can be used to SSH into the node. If the node was created with the `node_aws create` action, this should be the path to the private key file downloaded from the Amazon EC2 interface.

  Specify 'agent' if you have the key loaded in ssh-agent and available via the SSH_AUTH_SOCK variable.

  --login= | -l= | --username= - The name of the user Puppet should use when logging in to the node. This user should configured to allow passwordless access via the SSH key supplied in the -k -keyfile option.

  This is usually the root user.

  --pe-version= - Version of Puppet Enterprise to be passed to the installer script. Defaults to 1.1.

  --puppet-version= - Version of Puppet to be installed. This version is passed to the Puppet installer script.

--puppetagent-certname= - This option allows you to specify an optional puppet agent certificate name to configure on the target system. This option applies to the puppet-enterprise and puppet-enterprise-http installation scripts. If provided, this option will replace any puppet agent certificate name provided in the puppet enterprise answers file. This certificate name will show up in the console (or Puppet Dashboard) when the agent checks in for the first time.

- `save` - Invalid for this subcommand.: `SYNOPSIS`

  puppet node save [--terminus TERMINUS] [--extra HASH]

  `DESCRIPTION`

  Invalid for this subcommand.

- `search` - Invalid for this subcommand.: `SYNOPSIS`

  puppet node search [--terminus TERMINUS] [--extra HASH] query

  `DESCRIPTION`

  Invalid for this subcommand.

EXAMPLES

`classify`

Add the agent01.example.com node to the pe_agents group:

```
puppet node classify \
  --enc-server puppetmaster.example.com \
  --enc-port 3000 \
  --enc-ssl \
  --node-group pe_agents \
  agent01.example.com
```

`find`

Retrieve an "empty" (no classes, no ENC-imposed parameters, and an environment of "production") node:

$ puppet node find somenode.example.com --terminus plain --render-as yaml

Retrieve a node using the puppet master's configured ENC:

$ puppet node find somenode.example.com --terminus exec --mode master --render-as yaml

Retrieve the same node from the puppet master:

$ puppet node find somenode.example.com --terminus rest --render-as yaml

NOTES

This subcommand is an indirector face, which exposes `find`, `search`, `save`, and `destroy` actions for an indirected subsystem of Puppet. Valid termini for this face include:

- `active_record`
- `exec`
- `ldap`
- `memory`
- `plain`
- `rest`
- `store_configs`
- `yaml`

COPYRIGHT AND LICENSE

← Cloud Provisioning: Man Page: puppet node_aws — Index — Compliance: Basics and UI →

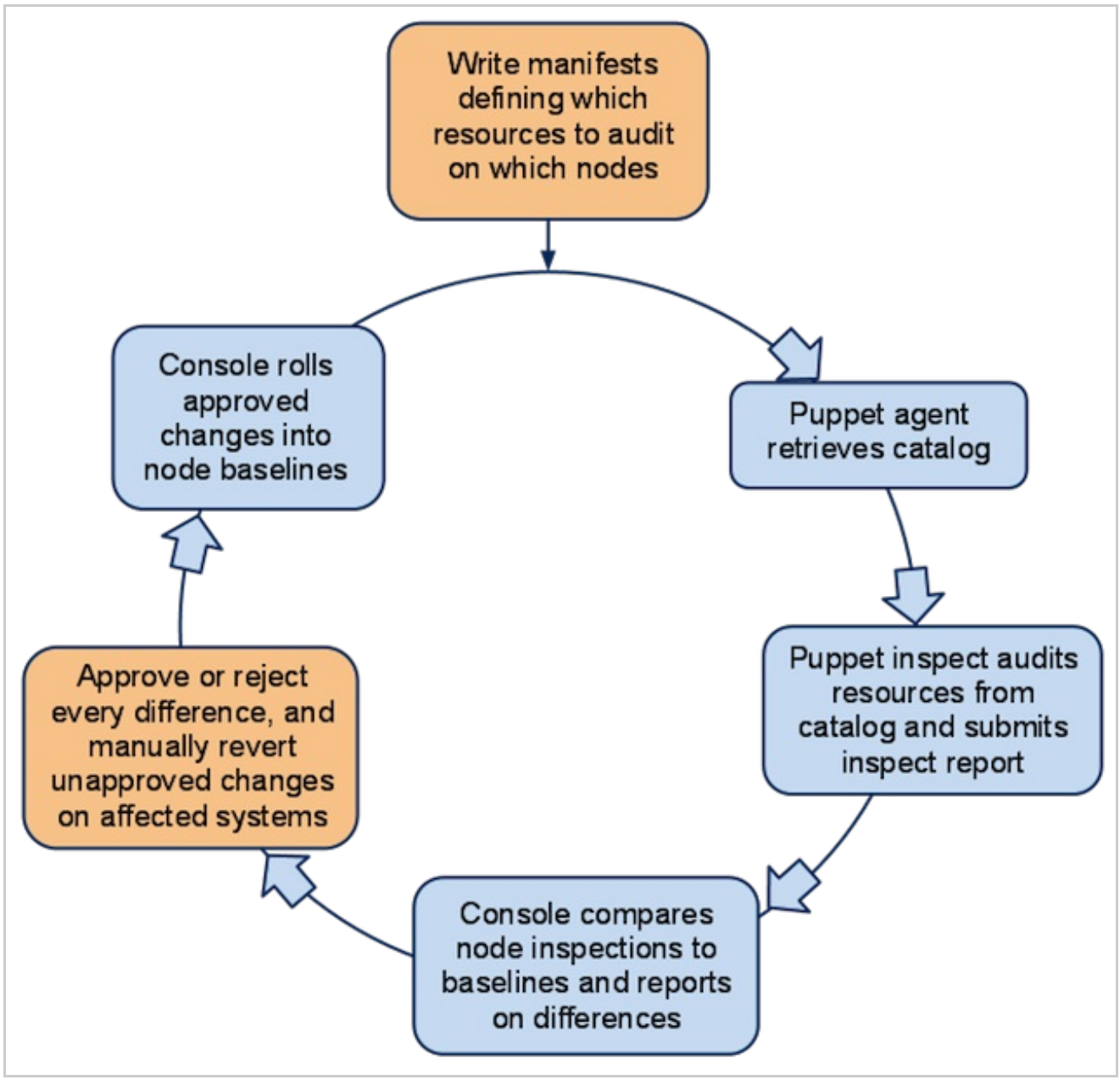
PE 2.0 » Compliance » Basics and UI

← Cloud Provisioning: Man Page: puppet node — Index — Compliance: Using the Compliance Workflow↦

Compliance Basics and UI

The Compliance Workflow Cycle

PE's compliance workflow is designed to audit changes in systems that are managed manually. It can also audit changes to resources being actively managed by Puppet.



- A sysadmin writes manifests defining which resources to audit on which nodes.
- Puppet agent retrieves and caches a catalog compiled from those manifests.
- Puppet inspect reads that catalog to discover which resources to audit, then submits an inspect report.
- The console analyzes inspect reports, then calculates a daily report of differences between the inspected system state and the approved baseline state. (Or, if this is the node's first inspect report, it will create an initial baseline for it.)

- A sysadmin uses the console's compliance pages to approve or reject every difference, then manually reverts any unapproved changes as necessary.
- The console then modifies the baseline to include any approved changes, and awaits the next day's inspect reports.

Concepts

Auditing

When using this workflow, Puppet audits the state of resources, rather than enforcing a desired state; it does not make changes to any audited resources. Instead, changes are to be made manually (or by some out-of-band process) and reviewed for approval after the fact.

After changes have been reviewed in the console, any approved changes will be considered the baseline state in future reports. Rejected changes will continue to be reported as non-baseline states until they are reverted manually on the affected machines.

Resources and Attributes

Any native Puppet resource type can be used in the baseline compliance workflow. As with similar compliance products, you can audit the content and metadata of files, but you can also audit user accounts, services, cron jobs, and anything for which a custom native type can be written.

Resources are audited by attribute — you can choose one or more attributes you wish to audit, or audit all attributes of that resource.

Compliance Manifests

The set of resources to audit is declared in standard Puppet manifests on the master and retrieved as a catalog by the agent. Instead of (or in addition to) declaring the desired state of a resource, these manifests should declare the `audit` metaparameter.

Inspect Reports

Each node being audited will routinely report the states of its audited resources. The documents it sends are called inspect reports, and differ from standard Puppet reports.

By default, every PE agent node sends daily inspect reports, regardless of whether it is auditing any resources.

Baselines

Conceptually, a baseline is a blessed inspect report for a single node: it lists the approved states for every audited resource on that node. Each node is associated with one and only one baseline, and nodes cannot share baselines. However, nodes with similar baselines can be grouped for convenience.

Baselines are maintained by the console. They change over time as administrators approve changes

to audited resources. Nodes reporting for the first time are assumed to be in a compliant state, and their first inspect report will become the baseline against which future changes are compared.

Groups

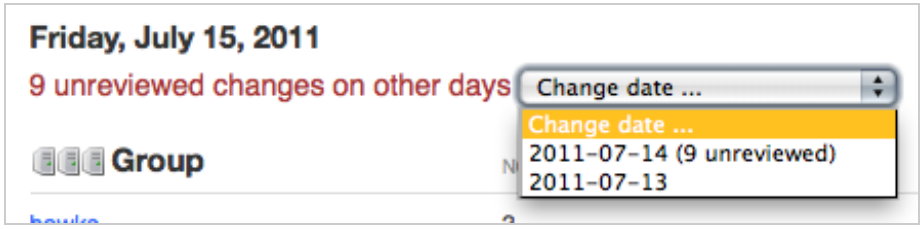
Although nodes cannot share baselines, nodes in groups can have similar changes approved or rejected en masse.

The Console's Compliance UI

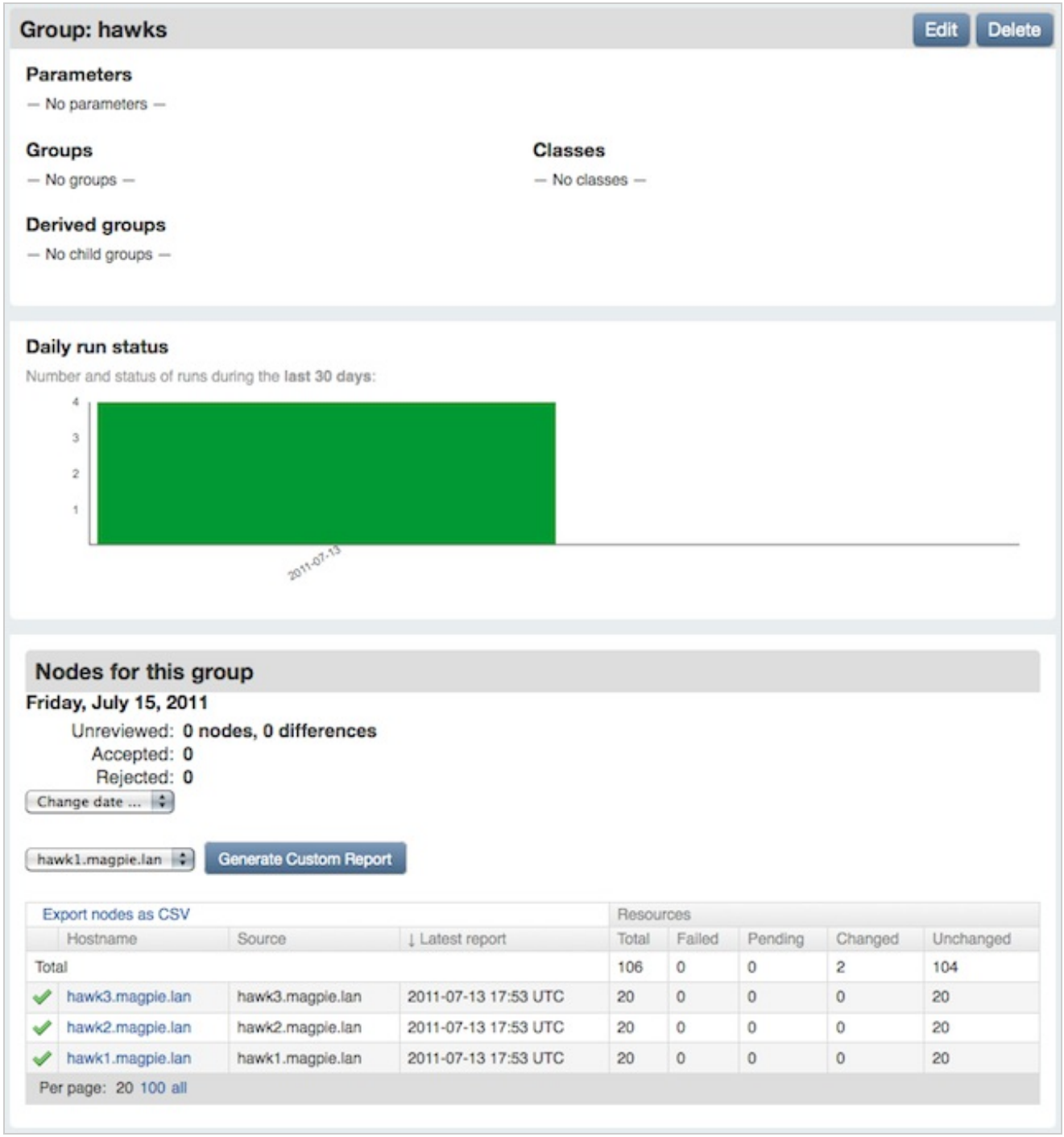
Puppet's compliance UI lives in the console, and consists of:

- A new summary and custom report control on each group's page
- A set of dedicated compliance pages, accessible from the "Compliance" link in the console's header menu

Each of these pages shows compliance information for a single day, and contains a date changer drop-down for changing the view. The most recently selected date will persist while you navigate these pages.



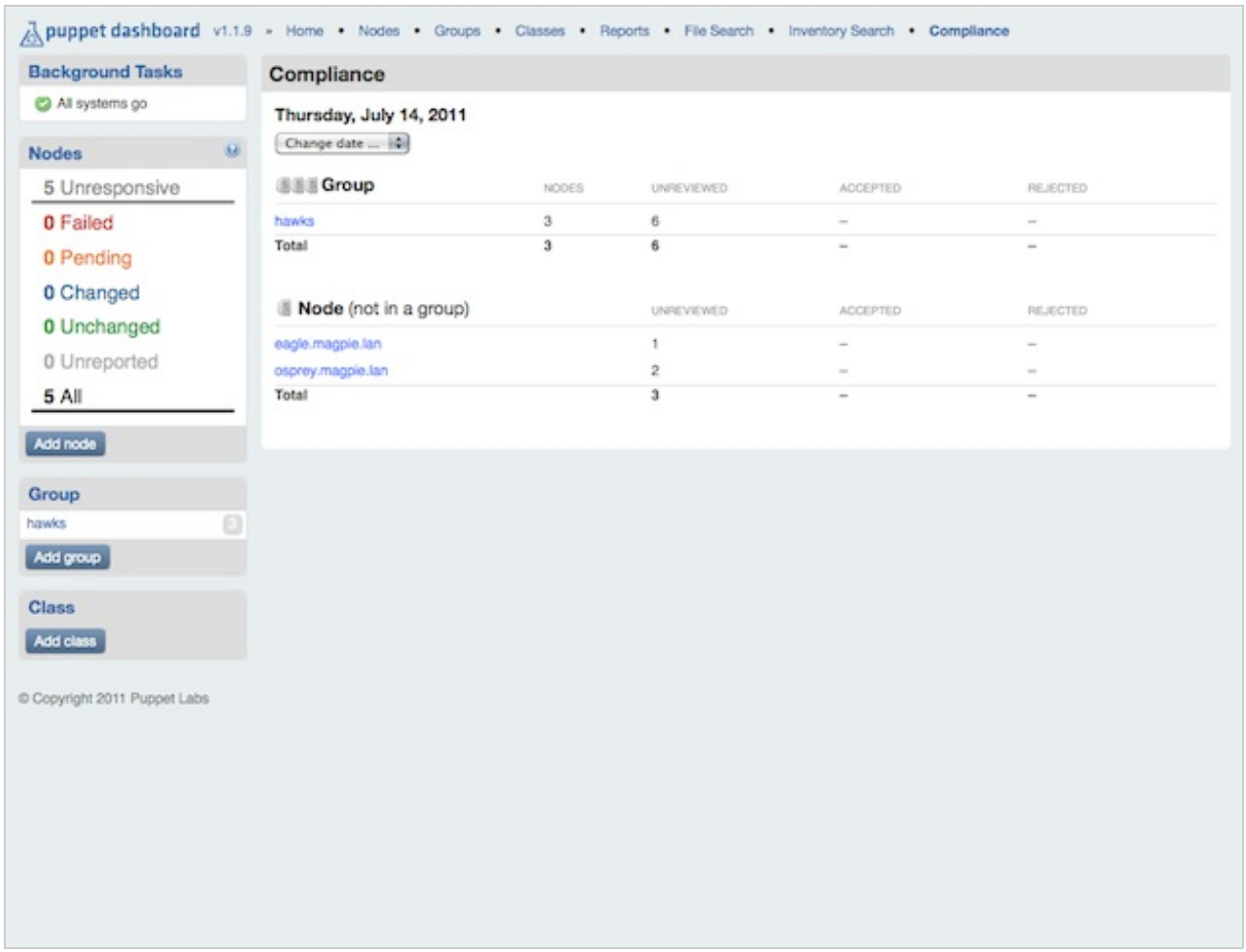
New Controls on Group Pages



Each group page will now have:

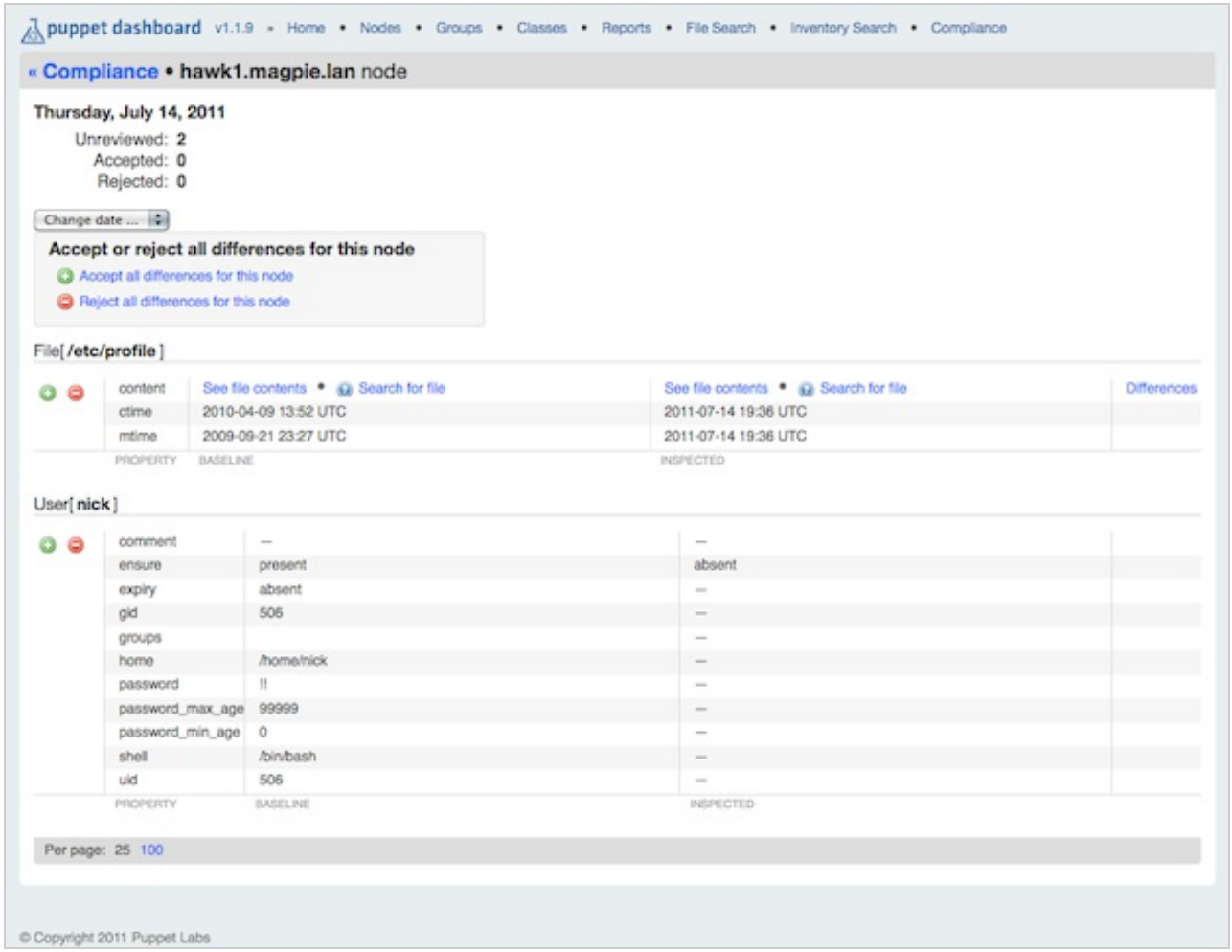
- A compliance summary in its node information section
- A control for generating custom reports

Compliance Overview



The main compliance overview page shows a single day's comparison results, with aggregate summaries for grouped nodes and individual summaries for groupless nodes.

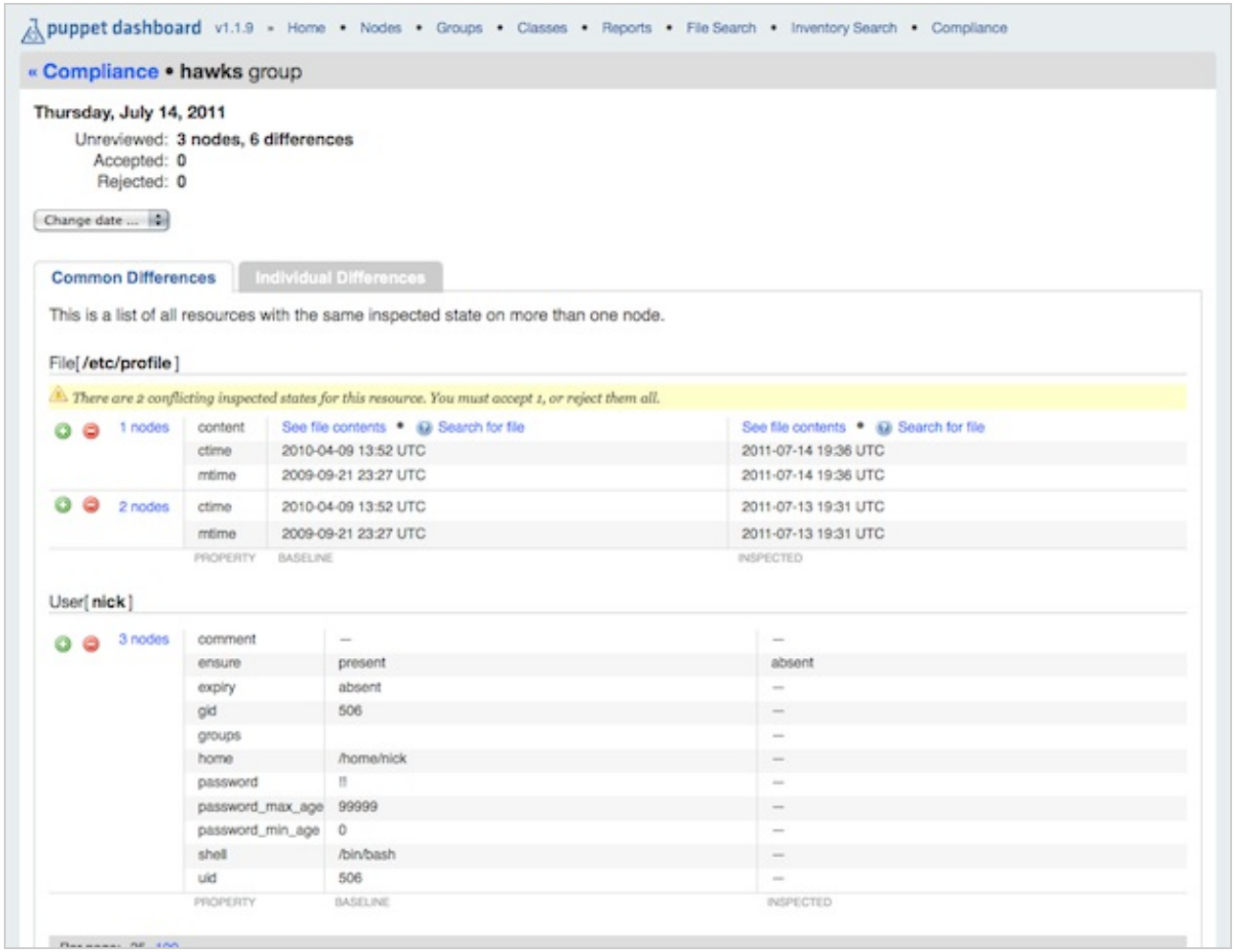
Compliance Node Page

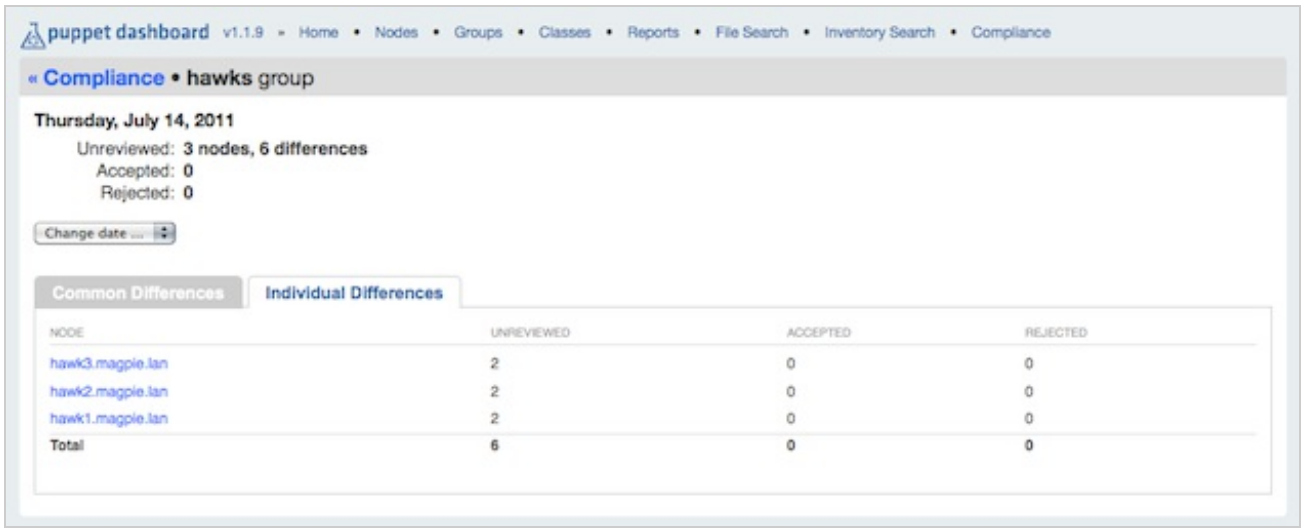


Individual node pages show one node's off-baseline inspection results for a single day. You can accept or reject changes from this page.

Links to the node pages of ungrouped nodes are displayed on the main compliance overview page. To see the details of a node which is in at least one group, navigate to its group page and use the "Individual Differences" tab.

Compliance Group Page





Compliance group pages show the collected differences for a group of nodes. Two tabs are available:

- Use the “Common Differences” tab to approve and reject aggregate changes en masse
- Use the “Individual Differences” tab to access node pages and act individually

Groups are one of the console’s core constructs, and nodes can be added to or removed from groups in the console’s main node pages.

Although the console allows groups to contain other groups, a compliance group page will only list nodes that are direct children of that group.

← Cloud Provisioning: Man Page: puppet node — Index — Compliance: Using the Compliance Workflow↦

PE 2.0 » Compliance » Using the Workflow

← Compliance: Basics and UI — Index — Compliance: Tutorial →

Using the Compliance Workflow

Prerequisites

Before using the compliance workflow, check to make sure that the `pe_compliance` Puppet class has been assigned to all of your agent nodes.

This probably does not require any action on your part; by default, `pe_compliance` is assigned to the default group, which all agent nodes are assigned to a few minutes after their first report.

Compliance Tasks

PE's compliance workflow consists of the following routine tasks:

- Writing compliance manifests
- Reviewing changes
- Comparing whole groups against a single baseline

See below for detailed explanations of these tasks.

Writing Compliance Manifests

To tell Puppet which resources to audit, you must write a collection of modules in the Puppet language and assign them to your nodes. See this manual's introduction to Puppet for a quick tutorial on creating a module and applying its classes.

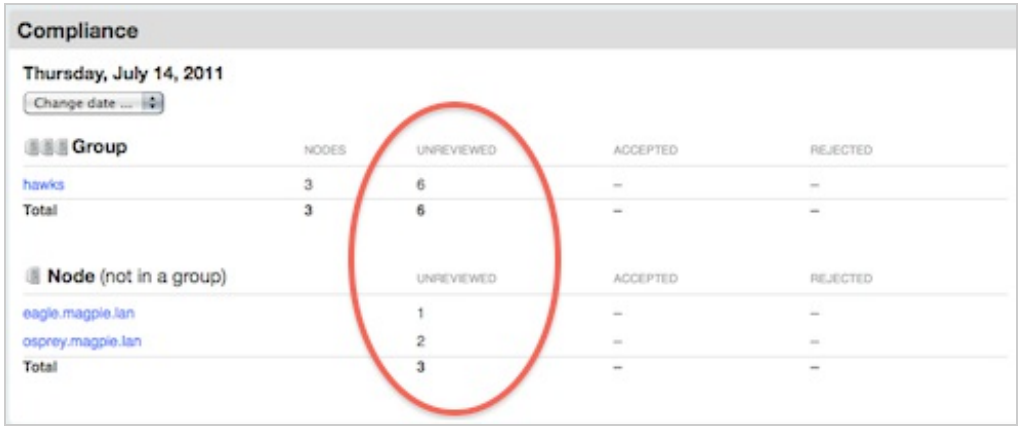
When writing compliance classes in Puppet, you should declare the `audit` metaparameter for each resource. The value of `audit` can be one attribute name, an array of attribute names, or `all`. Puppet can also actively manage attributes of an audited resource.

```
    file {'hosts':
      path  => '/etc/hosts',
      audit => 'content',
    }
    file {'/etc/sudoers':
      audit => [ensure, content, owner, group, mode, type],
    }
    user {'httpd':
      audit => 'all',
    }
    # Allow this user to change their password, but trigger an audit event when
they do:
    user {'admin':
      ensure => present,
      gid    => 'wheel',
      groups => ['admin'],
      shell  => '/bin/zsh',
```

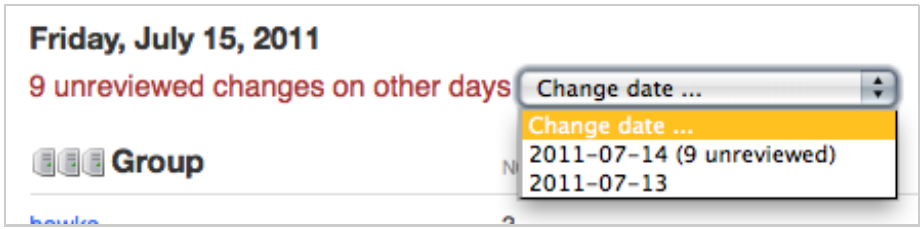
```
    audit  => 'password',
  }
```

Reviewing Changes

To audit changes, first scan the day's node and group summaries to see which nodes have unreviewed changes.



Once you've seen the overview, you can navigate to any pages with unreviewed changes. If there are any unreviewed changes on previous days, there will be a warning next to the date changer drop-down, and the drop-down will note which days aren't fully reviewed.

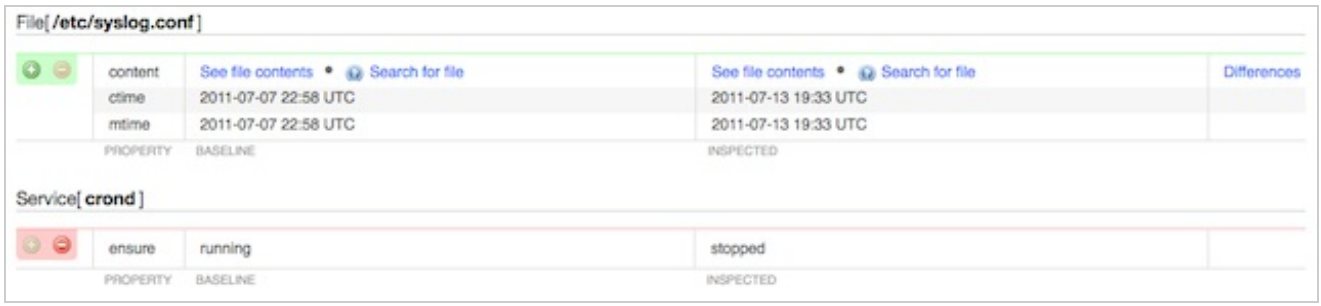


Changes can be accepted or rejected. Accepted changes will become part of the baseline for the next day's comparisons. Rejecting a change does nothing, and if an admin doesn't manually revert the change, it will appear as a difference again on the next day's comparisons.

When accepting multiple days' changes to the same resource, the most recently accepted change will win.

Reviewing Individual Nodes

Changes to individual nodes are straightforward to review: navigate to the node's page, view each change, and use the green plus and red minus buttons when you've decided whether the change was legitimate.



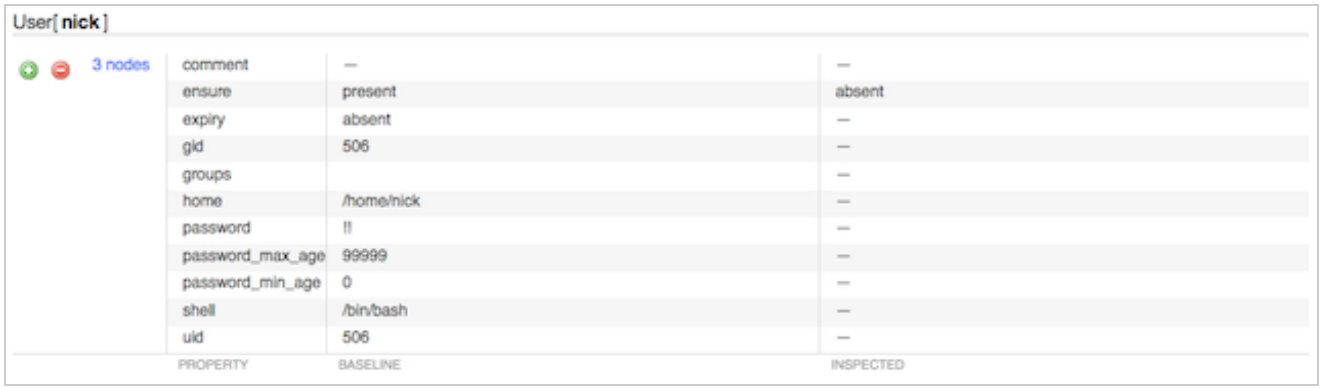
Each change is displayed in a table of attributes, with the previously approved (“baseline”) state on the left and the inspected state on the right. You will also see links for viewing the original and modified contents of any changed files, as well as a “differences” link for showing the exact□ changes.

You can also accept or reject all of the day’s changes to this node at once with the controls below the node’s summary. Note that rejecting all changes is “safe,” since it makes no edits to the node’s baseline; if you reject all changes without manually reverting them, you’re effectively deferring a□ decision on them to the next day.

Reviewing Groups

If you’ve collected similar nodes into groups in the console, you can greatly speed up the review of similar changes with the “Common Differences” tab. You can also use the “Individual Differences”□ tab to navigate to the individual nodes.

SAME CHANGE ON MULTIPLE NODES



If the same change was made on several nodes in a group, you can:

- Accept or reject the change for all affected nodes□
- View the individual node pages to approve or reject the change selectively

DIFFERENT CHANGES TO THE SAME RESOURCE

If different changes were made to the same resource on several nodes, the changes will be grouped for easy comparison. You can:

- Accept or reject each cluster of changes
- View the individual node pages to approve or reject the changes selectively

CONVERGENCE OF DIFFERING BASELINES

If several nodes in a group had a different baseline value for one resource but were recently brought into an identical state, you can click through to examine the previous baselines, and can:

- Approve or reject the single new state for all affected nodes
- View the individual node pages to approve or reject the changes selectively

Comparing Groups Against a Single Baseline

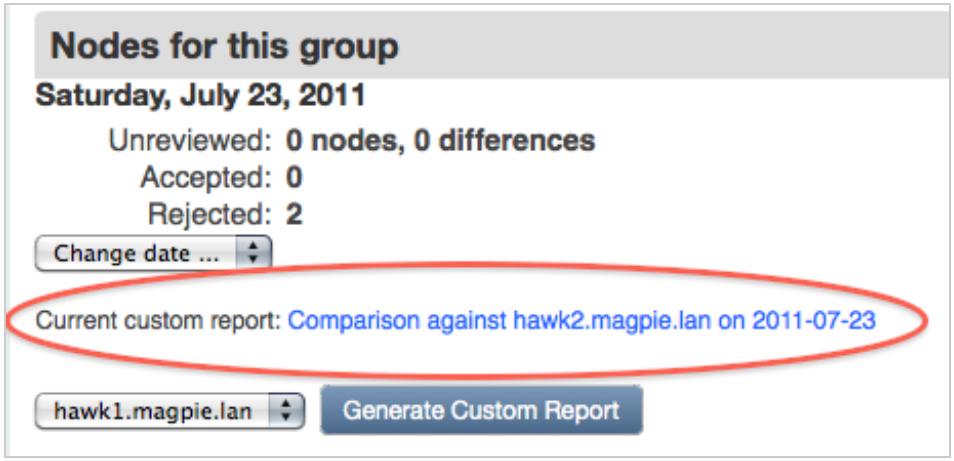
The console can also generate custom reports which compare an entire group to a single member node's baseline. While the day-to-day compliance views are meant for tracking changes to a node or group of nodes over time, custom reports are meant for tracking how far a group of nodes have drifted away from each other.

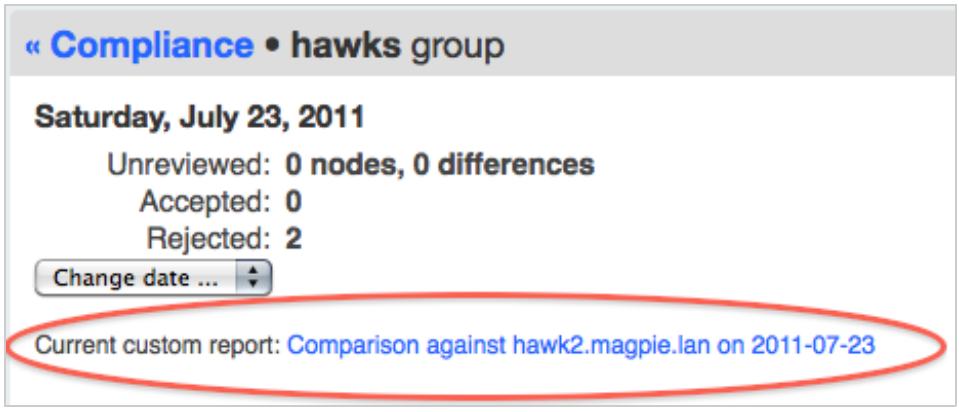
- Custom reports only examine the most recent inspection report for each group member
- Custom reports do not allow you to approve or reject changes

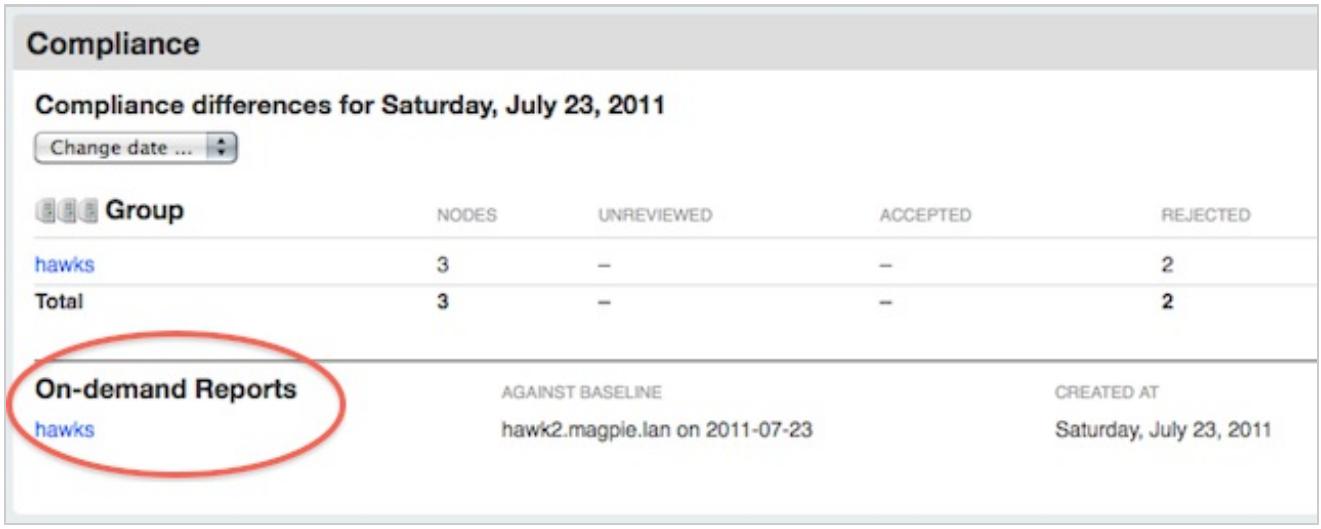
The console will maintain one cached custom report for each group; generating a new report for that group will erase the old one.

Custom reports are generated from the console's group pages — not from the group views in the compliance pages. To generate a report, choose which baseline to compare against and press the generate button; the report will be queued and a progress indicator will display. (The indicator is static markup that does not automatically poll for updates; you will need to reload the page periodically for updated status on the report.)

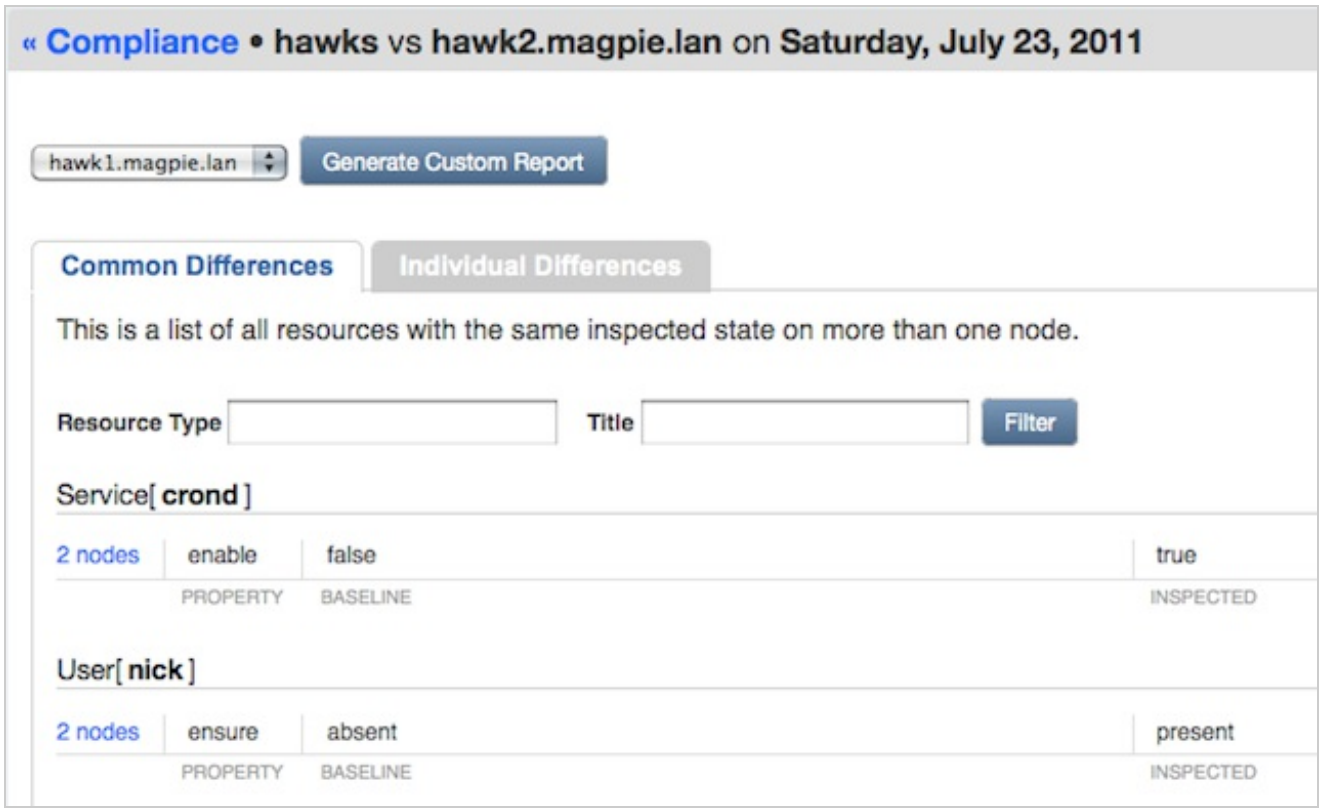
Once generated, custom reports can be viewed from the console's page for that group, the main compliance overview page, and the compliance group pages.







A custom report is split into a “Common Differences” tab and an “Individual Differences” tab. This is very similar to the layout of the group compliance review pages, and should be read in the same fashion; the only difference is that all comparisons are to a single baseline instead of per-node baselines.



← Compliance: Basics and UI — Index — Compliance: Tutorial →

PE 2.0 » Compliance » Tutorial

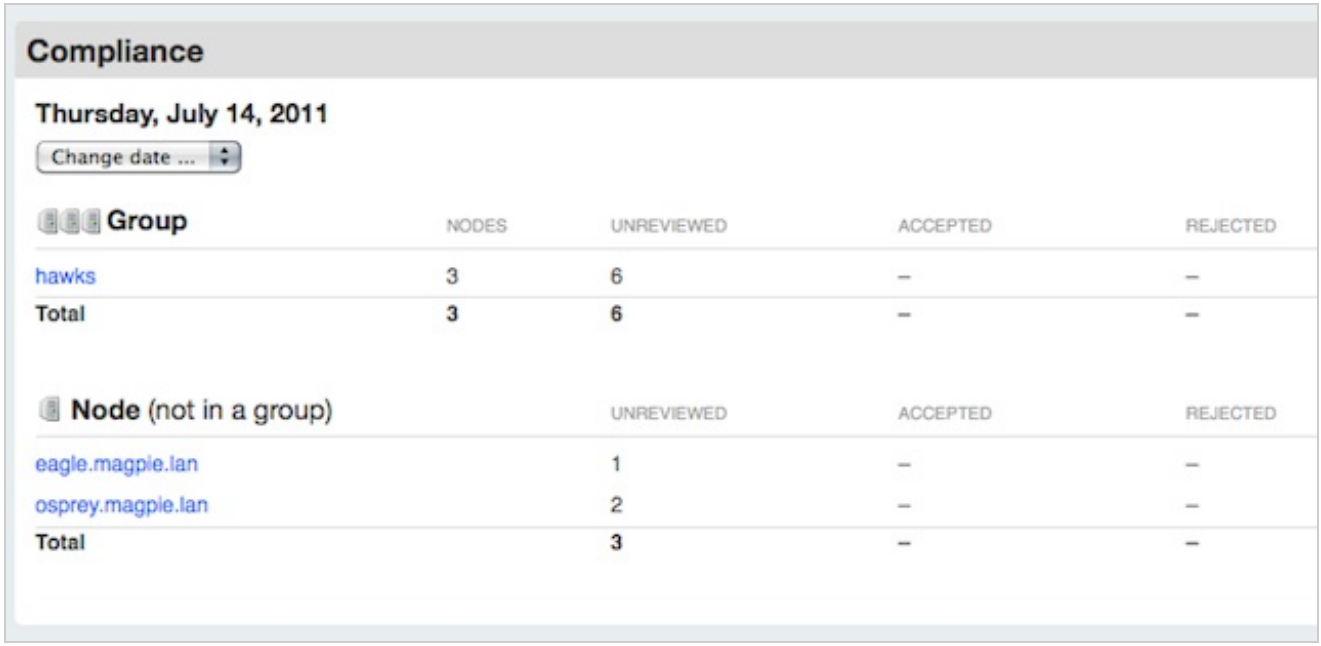
← Compliance: Using the Compliance Workflow — Index — PE Accounts Module: The `pe_accounts::user` Type →

Compliance Tutorial

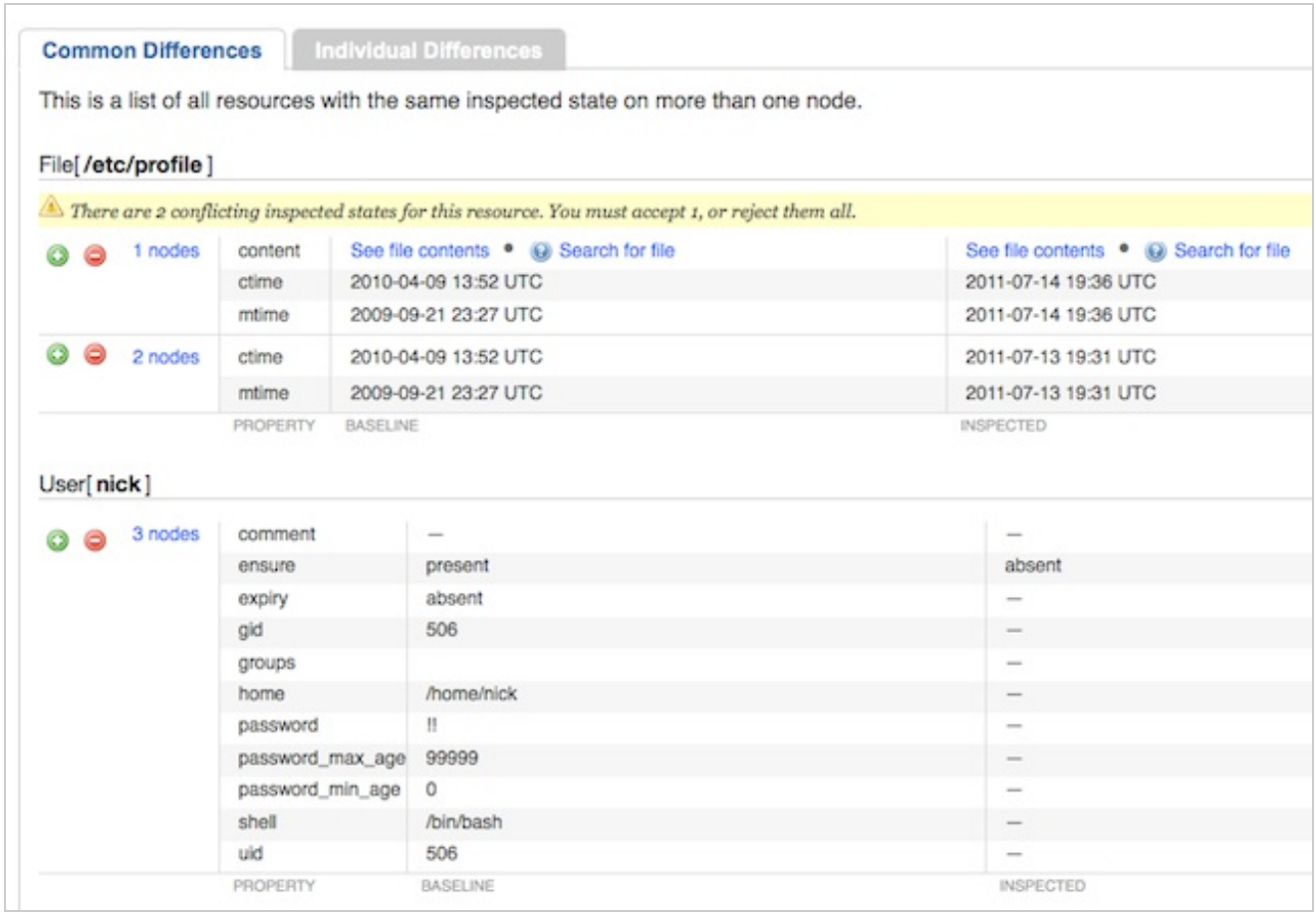
This brief walkthrough shows a compliance workflow auditing the state of the following resources:

- `File['/etc/profile']`
- `File['/etc/syslog.conf']`
- `Service['crond']`
- `Service['sshd']`
- `User['nick']`

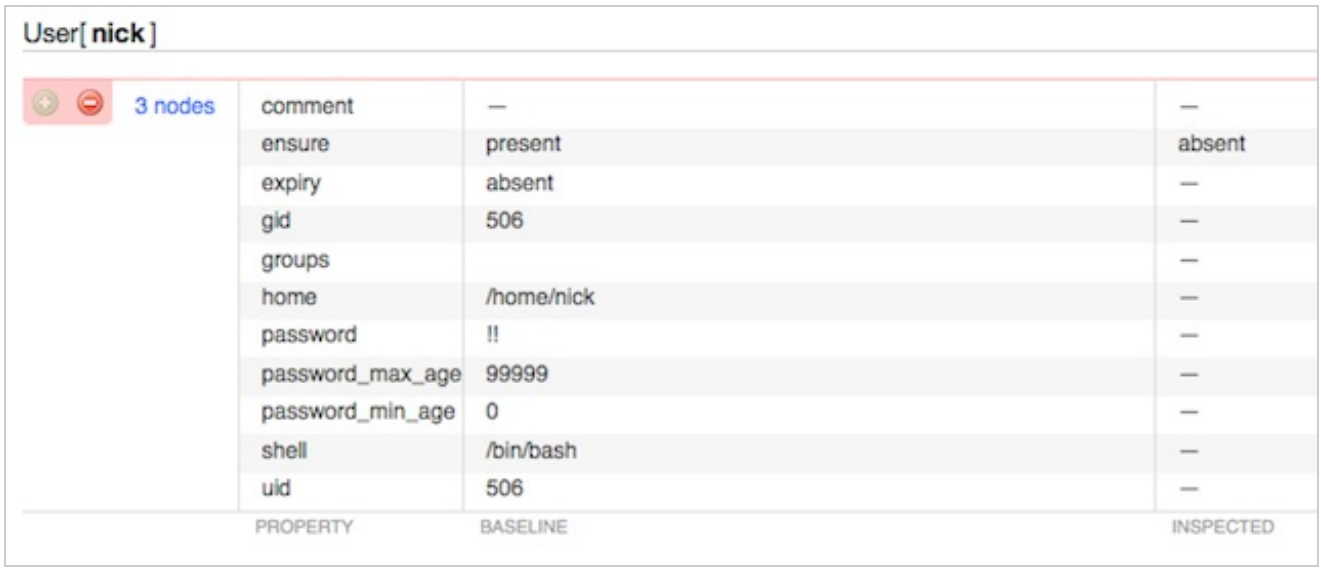
Morning, July 14, 2011



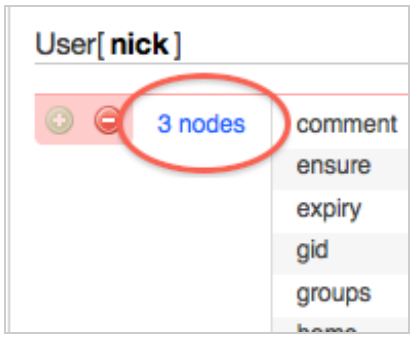
On Thursday morning, the admin notices unreviewed changes in a group of three nodes and a pair of ungrouped nodes. She checks the group first.



There, she notices that a user was completely deleted from all three nodes, and something odd happened with a file. She immediately rejects the deletion of the user…□

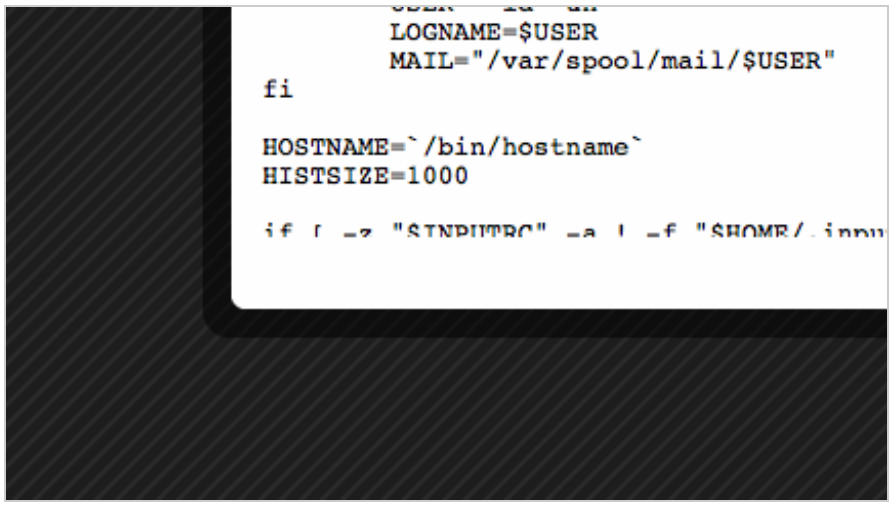


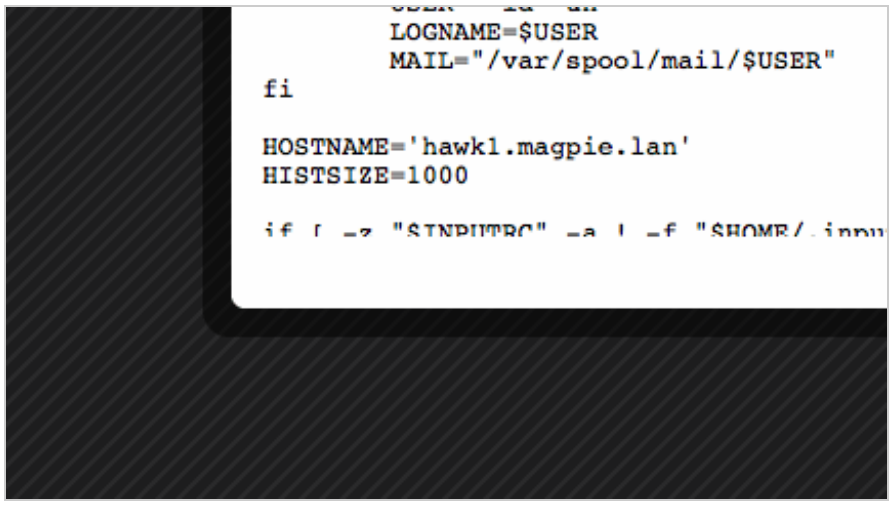
…and manually SSHes to the affected nodes to re-instate the account.□




```
[root@hawk1.example.com ~]# puppet resource group nick ensure=present gid=506
[root@hawk1.example.com ~]# puppet resource user nick ensure=present uid=506
gid=506
...
```

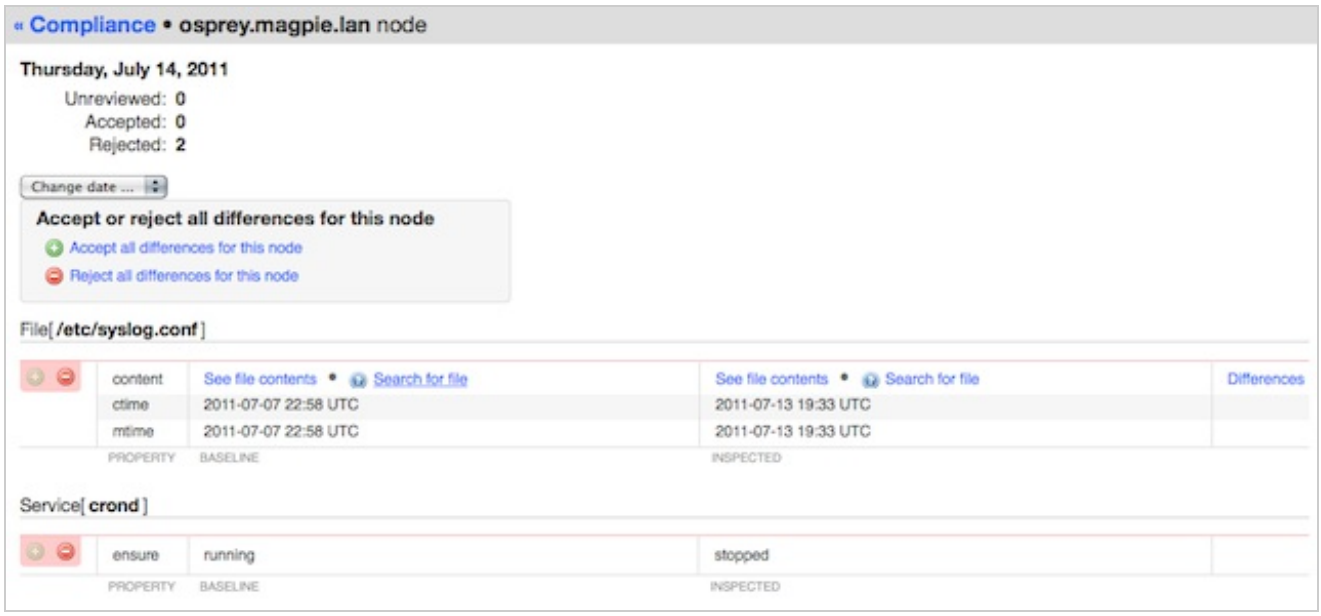
Then she takes a look at the file. It looks like two nodes had the ctime and mtime of the `/etc/profile` file changed, but no edits were made. This was probably nothing, but it piques her interest and she'll ask around about it later; in the meantime, she approves the change, since there's no functional difference. The other node, however, had its content changed. She drills down into the node view and checks the contents before and after:





That's not OK. It looks like someone was trying to resolve a DNS problem or something, but that's definitely not how she wants this machine configured. She rejects and manually reverts, and makes a note to find out what the problem they were trying to fix was.

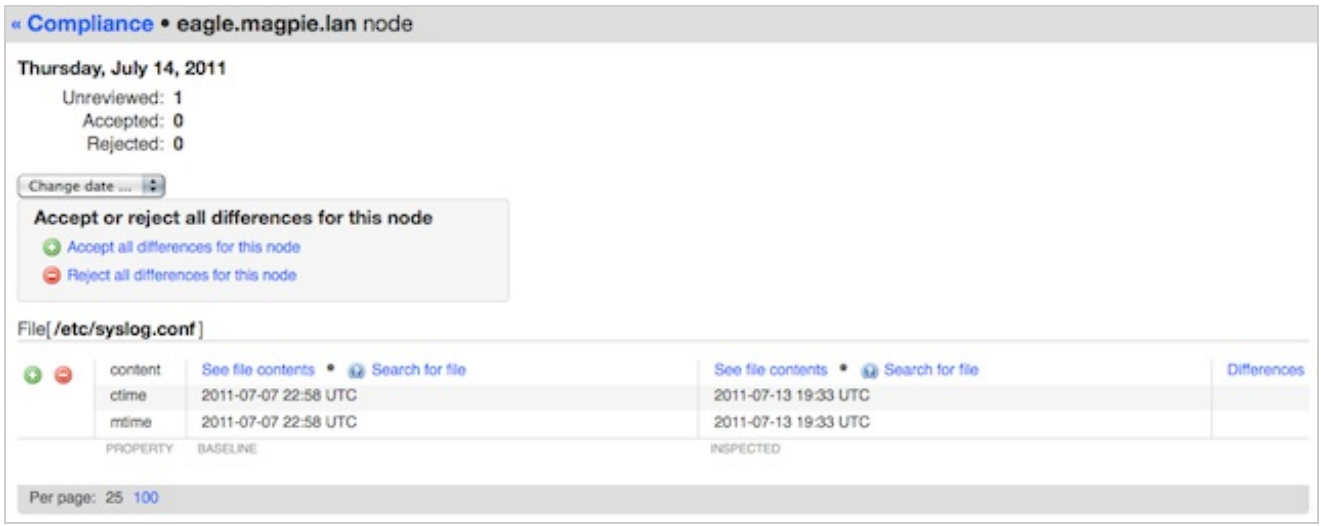
Next, the admin moves on to the individual nodes.

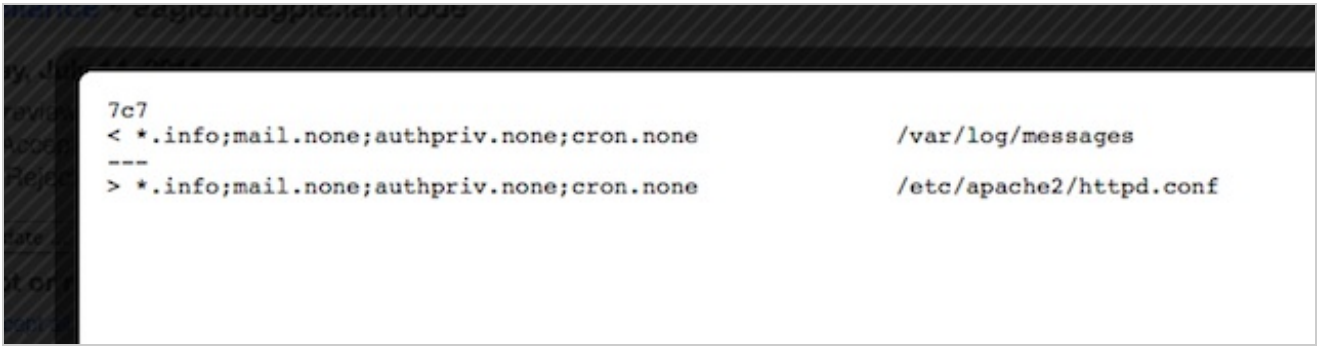


On the osprey server, something has stopped crond, which is definitely not good, and someone has made an edit to `/etc/syslog.conf`. She rejects the cron stoppage and restarts it, then checks the diff on the syslog config:

```
7c7
< *.info;mail.none;authpriv.none;cron.none  /var/log/messages
---
> *.info;mail.none;authpriv.none;cron.none  /etc/apache2/httpd.conf
```

That looks actively malicious. She rejects and reverts, then moves on to the eagle server to check on a hunch.





Yup: same dangerous change. She rejects and reverts, then spends the rest of her day figuring out how the servers got vandalized.

Morning, July 15, 2011

The next day, the admin's previous fixes to the syslog.conf and profile files on the various servers have caused changes to the ctime and mtime of those files. She approves her own changes, then decides that she should probably edit her manifests so that all but a select handful of file resources use `audit => [ensure, content, owner, group, mode, type]` instead of `audit => all`, in order to suppress otherwise meaningless audit events.

It's an otherwise uneventful day.

← Compliance: Using the Compliance Workflow — Index — PE Accounts Module: The `pe_accounts::user` Type →

PE 2.0 » Accounts Module » The pe_accounts::user Type

← Compliance: Tutorial — Index — PE Accounts Module: The `pe_accounts` Class →

The `pe_accounts::user` Type

This defined type is part of `pe_accounts`, a pre-built Puppet module that ships with Puppet Enterprise. It is available for use in your own manifests.

The `pe_accounts::user` type declares a user account. It offers several benefits over Puppet's core `user` type:

- It can create and manage the user's home directory as a Puppet resource.
- It creates and manages a primary group with the same name as the user, even on platforms where this is not the default.
- It can manage a set of SSH public keys for the user.
- It can easily lock the user's account, preventing all logins.

Puppet Enterprise uses this type internally to manage some of its own system users, but also exposes it as a public interface.

The `pe_accounts::user` type can be used on all of the platforms supported by Puppet Enterprise.

Note: In Puppet Enterprise 1.2, this type was called `accounts::user`; it was renamed in PE 2 to avoid namespace conflicts. If you are upgrading and wish to continue using the older name, the upgrader can install a wrapper module to enable it. See the chapter on upgrading for more details.

Usage Example

```
    # /etc/puppetlabs/puppet/modules/site/manifests/users.pp
    class site::users {
      # Declaring a dependency: we require several shared groups from the
site::groups class (see below).
      Class[site::groups] -> Class[site::users]

      # Setting resource defaults for user accounts:
      Pe_accounts::User {
        shell => '/bin/zsh',
      }

      # Declaring our pe_accounts::user resources:
      pe_accounts::user {'puppet':
        locked  => true,
        comment => 'Puppet Service Account',
        home    => '/var/lib/puppet',
        uid     => '52',
        gid     => '52',
      }
      pe_accounts::user {'sysop':
        locked  => false,
        comment => 'System Operator',
```

```
        uid     => '700',
        gid     => '700',
        groups  => ['admin', 'sudonopw'],
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
sysop+moduledevkey@puppetlabs.com'],
      }
      pe_accounts::user {'villain':
        locked  => true,
        comment => 'Test Locked Account',
        uid     => '701',
        gid     => '701',
        groups  => ['admin', 'sudonopw'],
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
villain+moduledevkey@puppetlabs.com'],
      }
      pe_accounts::user {'jeff':
        comment => 'Jeff McCune',
        groups => ['admin', 'sudonopw'],
        uid => '1112',
        gid => '1112',
        sshkeys => [
                    'ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
jeff+moduledevkey@puppetlabs.com',
                    'ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
jeff+moduledevkey2@puppetlabs.com',
                    'ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
jeff+moduledevkey3@puppetlabs.com',
                    'ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
jeff+moduledevkey4@puppetlabs.com'
                  ],
      }
      pe_accounts::user {'dan':
        comment => 'Dan Bode',
        uid => '1109',
        gid => '1109',
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
dan+moduledevkey@puppetlabs.com'],
      }
      pe_accounts::user {'nigel':
        comment => 'Nigel Kersten',
        uid => '2001',
        gid => '2001',
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
nigel+moduledevkey@puppetlabs.com'],
      }
    }

    # /etc/puppetlabs/puppet/modules/site/manifests/groups.pp
    class site::groups {
```

```
        # Shared groups:

        Group { ensure => present, }
        group {'developer':
          gid => '3003',
        }
        group {'sudonopw':
          gid => '3002',
        }
        group {'sudo':
          gid => '3001',
        }
        group {'admin':
          gid => '3000',
        }
      }
```

Parameters

Many of the type's parameters echo those of the standard `user` type.

`name`

The user's name. While limitations differ by operating system, it is generally a good idea to restrict user names to 8 characters, beginning with a letter. Defaults to the resource's title.

`ensure`

Whether the user and its primary group should exist. Valid values are `present` and `absent`. Defaults to `present`.

`shell`

The user's login shell. The shell must exist and be executable. Defaults to `/bin/bash`.

`comment`

A description of the user. Generally a user's full name. Defaults to the user's `name`.

`home`

The home directory of the user. Defaults to `/home/<user's name>`

`uid`

The user's uid number. Must be specified numerically; defaults to being automatically determined (`undef`).

`gid`

The gid of the primary group with the same name as the user; the `pe_accounts::user` type will create and manage this group. Must be specified numerically; defaults to being automatically determined (`undef`).

`groups`

An array of groups the user belongs to. The primary group should not be listed. Defaults to an empty array.

`membership`

Whether specified groups should be considered the complete list (`inclusive`) or the minimum list (`minimum`) of groups to which the user belongs. Valid values are `inclusive` and `minimum`; defaults to `minimum`.

`password`

The user's password, in whatever encrypted format the local machine requires. Be sure to enclose any value that includes a dollar sign ($) in single quotes (‘). Defaults to `'!!'`, which prevents the user from logging in with a password.

`locked`

Whether the user should be prevented from logging in; set this to `true` for system users and users whose login privileges have been revoked. Valid values are `true` and `false`; defaults to false.

`sshkeys`

An array of SSH public keys associated with the user. Unlike with the `ssh_authorized_key` type, these should be complete public key strings that include the type and name of the key, exactly as the key would appear in its `id_rsa.pub` or `id_dsa.pub` file. Defaults to an empty array.

← Compliance: Tutorial — Index — PE Accounts Module: The `pe_accounts` Class →

PE 2.0 » Accounts Module » The pe_accounts Class

← PE Accounts Module: The `pe_accounts::user` Type — Index — Maintenance: Common Configuration Errors →

The `pe_accounts` Class

This class is part of `pe_accounts`, a pre-built Puppet module that ships with Puppet Enterprise.

The `pe_accounts` class can do any or all of the following:

- Create and manage a set of `pe_accounts::user` resources
- Create and manage a set of shared `group` resources
- Maintain a pair of rules in the `sudoers` file to grant privileges to the `sudo` and `sudonopw` groups

This class is designed for cases where your account data is maintained separately from your Puppet manifests. This usually means one of the following is true:

- The data is being read from a non-Puppet directory service or CMDB, probably with a custom function.
- The data is being maintained manually by a user who does not write Puppet code.
- The data is being generated by an out-of-band process.

If your site's account data will be maintained manually by a sysadmin able to write Puppet code, it will make more sense to maintain it as a normal set of `pe_accounts::user` and `group` resources, although you may still wish to use the `pe_accounts` class to maintain `sudoers` rules.

To manage users and groups with the `pe_accounts` class, you must prepare a data store and configure the class for the data store when you declare it.

Note: In Puppet Enterprise 1.2, this class was called `accounts`; it was renamed in PE 2 to avoid namespace conflicts. If you are upgrading and wish to continue using the older name, the upgrader can install a wrapper module to enable it. See the chapter on upgrading for more details.

Note: In Puppet Enterprise 2.0.0, this class is assigned to the console's default group with no parameters, which will prevent it from being redeclared with any configuration. To use the class, you must:

- Unassign it from the default group in the console
- Create a wrapper module that declares this class with the necessary parameters
- Re-assign the wrapper class to whichever nodes need it

Usage Example

To use YAML files as a data store:

```
class {'pe_accounts':
  data_store => yaml,
```

```
}
```

To use a Puppet class as a data store (and manage `sudoers` rules):

```
class {'pe_accounts':
  data_store     => namespace,
  data_namespace => 'site::pe_accounts::data',
  manage_sudoers => true,
}
```

To manage `sudoers` rules without managing any users or groups:

```
class {'pe_accounts':
  manage_users   => false,
  manage_groups  => false,
  manage_sudoers => true,
}
```

Data Stores

Account data can come from one of two sources: a Puppet class that declares three variables, or a set of three YAML files stored in `/etc/puppetlabs/puppet/data`.

Using a Puppet Class as a Data Store

This option is most useful if you are able to generate or import your user data with a custom function, which may be querying from an LDAP directory or some other arbitrary data source.

The Puppet class containing the data must have a name ending in `::data`. (We recommend `site::pe_accounts::data`.) This class must declare the following variables:

- `$users_hash` should be a hash in which each key is the title of a `pe_accounts::user` resource and each value is a hash containing that resource's attributes and values.
- `$groups_hash` should be a hash in which each key is the title of a group and each value is a hash containing that resource's attributes and values.

See below for examples of the data formats used in these variables.

When declaring the `pe_accounts` class to use data in a Puppet class, use the following attributes:

```
data_store     => namespace,
data_namespace => {name of class},
```

Using YAML Files as a Data Store

This option is most useful if your user data is being generated by an out-of-band process or is being maintained by a user who does not write Puppet manifests.

When storing data in YAML, the following valid YAML files must exist in `/etc/puppetlabs/puppet/data`:

- `pe_accounts_users_hash.yaml`, which should contain an anonymous hash in which each key is the title of a `pe_accounts::user` resource and each value is a hash containing that resource's attributes and values.
- `pe_accounts_groups_hash.yaml`, which should contain an anonymous hash in which each key is the title of a group and each value is a hash containing that resource's attributes and values.

See below for examples of the data formats used in these variables.

When declaring the `pe_accounts` class to use data in YAML files, use the following attribute:

```
data_store => yaml,
```

Data Formats

This class uses three hashes of data to construct the `pe_accounts::user` and `group` resources it manages.

THE USERS HASH

The users hash represents a set of `pe_accounts::user` resources. Each key should be the title of a `pe_accounts::user` resource, and each value should be another hash containing that resource's attributes and values.

PUPPET EXAMPLE

```
    $users_hash = {
      sysop => {
        locked  => false,
        comment => 'System Operator',
        uid     => '700',
        gid     => '700',
        groups  => ['admin', 'sudonopw'],
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
sysop+moduledevkey@puppetlabs.com'],
      },
      villain => {
        locked  => true,
        comment => 'Test Locked Account',
        uid     => '701',
        gid     => '701',
        groups  => ['admin', 'sudonopw'],
        sshkeys => ['ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
```

```
  villain+moduledevkey@puppetlabs.com'],
      },
    }
```

YAML EXAMPLE

```
---
sysop:
  locked: false
  comment: System Operator
  uid: '700'
  gid: '700'
  groups:
  - admin
  - sudonopw
  sshkeys:
  - ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
sysop+moduledevkey@puppetlabs.com
villain:
  locked: true
  comment: Test Locked Account
  uid: '701'
  gid: '701'
  groups:
  - admin
  - sudonopw
  sshkeys:
  - ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAwLBhQefRiXHSbVNZYKu2o8VWJjZJ/B4LqICXuxhiiNSCmL8j+5zE/V
villain+moduledevkey@puppetlabs.com
```

THE GROUPS HASH

The groups hash represents a set of shared `group` resources. Each key should be the title of a `group` resource, and each value should be another hash containing that resource's attributes and values.

PUPPET EXAMPLE

```
    $groups_hash = {
      developer => {
        gid    => 3003,
        ensure => present,
      },
      sudonopw => {
        gid    => 3002,
        ensure => present,
      },
      sudo     => {
        gid    => 3001,
        ensure => present,
      },
```

```
        admin    => {
          gid    => 3000,
          ensure => present,
        },
      }
```

YAML EXAMPLE

```
---
developer:
 gid: "3003"
 ensure: "present"
sudonopw:
 gid: "3002"
 ensure: "present"
sudo:
 gid: "3001"
 ensure: "present"
admin:
 gid: "3000"
 ensure: "present"
```

Parameters

`manage_groups`

Whether to manage a set of shared groups, which can be used by all `pe_accounts::user` resources. If true, your data store must define these groups in the `$groups_hash` variable or the `pe_accounts_groups_hash.yaml` file. Allowed values are `true` and `false`; defaults to `true`.

`manage_users`

Whether to manage a set of `pe_accounts::user` resources. If true, your data store must define these users in the `$users_hash` variable or the `pe_accounts_users_hash.yaml` file. Allowed values are `true` and `false`; defaults to `true`.

`manage_sudoers`

Whether to add sudo rules to the node's `sudoers` file. If true, the class will add `%sudo` and `%sudonopw` groups to the `sudoers` file and give them full sudo and passwordless sudo privileges respectively. You will need to make sure that the `sudo` and `sudonopw` groups exist in the groups hash, and that your chosen users have those groups in their `groups` arrays. Managing `sudoers` is not supported on Solaris.

Allowed values are `true` and `false`; defaults to `false`.

`data_store`

Which data store to use for accounts and groups.

When set to `namespace`, data will be read from the puppet class specified in the `data_namespace` parameter. When set to `yaml`, data will be read from specially-named YAML files in the `/etc/puppetlabs/puppet/data` directory. (If you have changed your `$confdir`, it will look in `$confdir/data`.) Example YAML files are provided in the `ext/data/` directory of this module.

Allowed values are `yaml` and `namespace`; defaults to `namespace`.

`data_namespace`

The Puppet namespace from which to read data. This must be the name of a Puppet class, and must end with `::data` (we recommend using `site::pe_accounts::data`); the class will automatically be declared by the `pe_accounts` class. The class cannot have any parameters, and must declare variables named:

- `$users_hash`
- `$groups_hash`

See the `pe_accounts::data` class included in this module (in `manifests/data.pp`) for an example; see the data formats section for information on each hash's data structure.

Defaults to `pe_accounts::data`.

`sudoers_path`

The path to the `sudoers` file on this system. Defaults to `/etc/sudoers`.

← PE Accounts Module: The `pe_accounts::user` Type — Index — Maintenance: Common Configuration Errors →

PE 2.0 » Maintenance and Troubleshooting » Common Config Errors

---

← PE Accounts Module: The `pe_accounts` Class — Index — Maintenance: Console Maintenance Tasks →

Troubleshooting Common Configuration Errors

The Installer is Failing!

Here are the main problems that can cause an install to blow up.

Is DNS Wrong?

If name resolution at your site isn't quite behaving right, PE's installer can go haywire.

- Puppet agent has to be able to reach the puppet master server at one of its valid DNS names. (Specifically, the name you identified as the master's hostname during the installer interview.)
- The puppet master also has to be able to reach itself at the puppet master hostname you chose during installation.
- If you've split the master and console roles onto different servers, they have to be able to talk to each other as well.

Are the Security Settings Wrong?

The installer fails in a similar way when the system's firewall or security group is restricting the ports Puppet uses.

- Puppet communicates on ports 8140, 61613, and 443. If you are installing the puppet master and the console on the same server, it must accept inbound traffic on all three ports. If you've split the two roles, the master must accept inbound traffic on 8140 and 61613, and the console must accept inbound traffic on 8140 and 443.
- If your puppet master has multiple network interfaces, make sure it is allowing traffic via the IP address that its valid DNS names resolve to, not just via an internal interface.

Did You Try to Install the Console Before the Puppet Master?

If you are installing the console and the puppet master on separate servers and tried to install the console first, the installer may fail.

How Do I Recover From a Failed Install?

First, fix any configuration problem that may have caused the install to fail; see above for a list of the most common errors.

Next, download, move, and run the uninstaller script. (This script was not included in Puppet Enterprise 2.0, but will be included in all future releases.)

- Click here to download the uninstaller, or use `curl` or `wget` to download it directly to the target machine.
- Copy the uninstaller to the target machine, and move it into the directory which contains the installer script. The uninstaller and the installer must be in the same directory.

- Make the uninstaller executable, then run it:

```
# sudo chmod +x puppet-enterprise-uninstaller
# sudo ./puppet-enterprise-uninstaller
```

After you have run the uninstaller, you can safely run the installer again.

Agent Nodes Can't Retrieve Their Configurations!

Is the Puppet Master Reachable From the Agents?

Although this would probably have caused a problem during installation, it's worth checking it first.
You can check whether the master is reachable and active by trying:

```
$ telnet <puppet master's hostname> 8140
```

If the puppet master is alive and reachable, you'll get something like:

```
Trying 172.16.158.132...
Connected to screech.example.com.
Escape character is '^]'.
```

Otherwise, it will return something like "nodename nor servname provided, or not known."

To fix this, make sure the puppet master server is reachable at the DNS name your agents know it by, and make sure that the `pe-httpd` service is running.

Can the Puppet Master Reach the Console?

The puppet master depends on the console for the names of the classes an agent node should get. If it can't reach the console, it can't compile configurations for nodes.

Check the puppet agent logs on your nodes, or run `puppet agent --test` on one of them; if you see something like `err: Could not retrieve catalog from remote server: Error 400 on SERVER: Could not find node 'agent01.example.com'; cannot compile`, the master may be failing to find the console.

To fix this, make sure that the console is alive by navigating to its web interface. If it can't be reached, make sure DNS is set up correctly for the console server, and ensure that the `pe-httpd` service on it is running.

If the console is alive and reachable from the master but the master can't retrieve node info from it, the master may be configured with the wrong console hostname. You'll need to:

- Edit the `reporturl` setting in the master's `/etc/puppetlabs/puppet/puppet.conf` file to point

to the correct host.

- Edit the `ENC_BASE_URL` variable in the master's `/etc/puppetlabs/puppet-dashboard/external_node` file to point to the correct host.

Do Your Agents Have Signed Certificates?

Check the puppet agent logs on your nodes and look for something like the following:

```
warning: peer certificate won't be verified in this SSL session
```

If you see this, it means the agent has submitted a certificate signing request which hasn't yet been signed. Run `puppet cert list` on the puppet master to see a list of pending requests, then run `puppet cert sign <NODE NAME>` to sign a given node's certificate. The node should successfully retrieve and apply its configuration the next time it runs.

Do Agents Trust the Master's Certificate?

Check the puppet agent logs on your nodes and look for something like the following:

```
err: Could not retrieve catalog from remote server: 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 could be one of several things.

ARE AGENTS CONTACTING THE MASTER AT A VALID DNS NAME?

When you installed the puppet master role, you approved a list of valid DNS names to be included in the master's certificate. Agents will ONLY trust the master if they contact it at one of THESE hostnames.

To check the hostname agents are contacting the master at, run `puppet agent --configprint server`. If this is not one of the valid DNS names you chose during installation of the master, edit the `server` setting in the agents' `/etc/puppetlabs/puppet/puppet.conf` files to point to a valid DNS name.

If you need to reset your puppet master's valid DNS names, run the following:

```
$ /etc/init.d/pe-httpd stop
$ puppet cert clean <puppet master's certname>
$ puppet cert generate <puppet master's certname> --dns_alt_names=<comma-
separated list of DNS names>
$ /etc/init.d/pe-httpd start
```

IS TIME IN SYNC ON YOUR NODES?

…and was time in sync when your certificates were created?

Compare the output of `date` on your nodes. Then, run the following command on the puppet master to check the validity dates of a given certificate:

```
$ openssl x509 -text -noout -in $(puppet master --configprint
ssldir)/certs/<NODE NAME>.pem
```

- If time is out of sync, get it in sync. Keep in mind that NTP can behave unreliably on virtual machines.
- If you have any certificates that aren't valid until the future…
  - Delete the certificate on the puppet master with `puppet cert clean <NODE NAME>`.
  - Delete the SSL directory on the offending agent with `rm -rf $(puppet agent --configprint ssldir)`.
  - Run `puppet agent --test` on that agent to generate a new certificate request, then sign that request on the master with `puppet cert sign <NODE NAME>`.

DID YOU PREVIOUSLY HAVE AN UNRELATED NODE WITH THE SAME CERTNAME?

If a node re-uses an old node's certname and the master retains the previous node's certificate, the new node will be unable to request a new certificate.

Run the following on the master:

```
$ puppet cert clean <NODE NAME>
```

Then, run the following on the agent node:

```
$ rm -rf $(puppet agent --configprint ssldir)
$ puppet agent --test
```

This should properly generate a new signing request.

Can Agents Reach the Filebucket Server?

Agents attempt to back up files to the puppet master, but they get the filebucket hostname from the site manifest instead of their configuration file. If puppet agent is logging "could not back up" errors, your nodes are probably trying to back up files to the wrong hostname. These errors look like this:

```
err:
/Stage[main]/Pe_mcollective/File[/etc/puppetlabs/mcollective/server.cfg]/content:

change from {md5}778087871f76ce08be02a672b1c48bdc to
```

```
{md5}e33a27e4b9a87bb17a2bdff115c4b080 failed: Could not back up
/etc/puppetlabs/mcollective/server.cfg: getaddrinfo: Name or service not known
```

This usually happens when puppet master is installed with a certname that isn't its hostname. To fix these errors, edit `/etc/puppetlabs/puppet/manifests/site.pp` on the puppet master so that the following resource's `server` attribute points to the correct hostname:

```
    # Define filebucket 'main':
    filebucket { 'main':
      server => '<PUPPET MASTER'S DNS NAME>',
      path   => false,
    }
```

Changing this on the puppet master will fix the error on all agent nodes.

`node_vmware` and `node_aws` Aren't Working!

If the cloud provisioning actions are failing with an "err: Missing required arguments" message, you need to create a `~/.fog` file and populate it with the appropriate credentials.

← PE Accounts Module: The `pe_accounts` Class — Index — Maintenance: Console Maintenance Tasks →

PE 2.0 » Maintenance and Troubleshooting » Console Maintenance Tasks

---

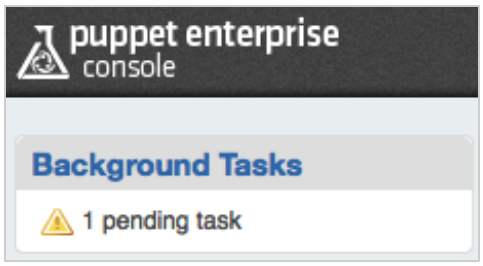
← Maintenance: Common Configuration Errors — Index — Maintenance: Installing Additional Components →

Console Maintenance Tasks

If PE's console becomes sluggish or begins taking up too much space on disk, there are several maintenance tasks that can improve its performance.

Restarting the Background Tasks

The console uses several worker processes to process reports in the background, and it displays a running count of pending tasks in the upper left corner of the interface:



If the number of pending tasks appears to be growing linearly, it is possible the background task processes died and left invalid PID files. To restart the worker tasks, run:

```
$ sudo /etc/init.d/pe-puppet-dashboard-workers restart
```

The number of pending tasks shown in the console should start decreasing rapidly after restarting the workers.

Optimizing the Database

Since the console turns over a lot of data, you can improve its speed and disk consumption by periodically optimizing its MySQL database with the `db:raw:optimize` task.

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
db:raw:optimize
```

If you find you need to optimize the database, we recommend that you do so with a monthly cron job.

Cleaning Old Reports

Agent node reports will build up over time in the console's database. If you wish to delete the oldest reports, for performance, storage, or policy reasons, you can use the `reports:prune` rake task.

For example, to delete reports more than one month old:

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
reports:prune upto=1 unit=mon
```

Although this task should be run regularly as a cron job, the frequency with which it should be run will depend on your site's policies.

If you run the `reports:prune` task without any arguments, it will display further usage instructions. The available units of time are `mon`, `yr`, `day`, `min`, `wk`, and `hr`.

Database backups

Although you can back up and restore the console's database with any standard MySQL tools, the `db:raw:dump` and `db:raw:restore` rake tasks can simplify the process.

Dumping the Database

To dump the console's `production` database to a file called `production.sql`:

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
db:raw:dump
```

Or dump it to a specific file:

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
FILE=/my/backup/file.sql db:raw:dump
```

Restoring the Database

To restore the console's database from a file called `production.sql` to your `production` environment:

```
$ sudo /opt/puppet/bin/rake \
-f /opt/puppet/share/puppet-dashboard/Rakefile \
RAILS_ENV=production \
FILE=production.sql db:raw:restore
```

PE 2.0 » Maintenance and Troubleshooting » Installing Additional Components

← Maintenance: Console Maintenance Tasks — Index — Maintenance: Reconfiguring Complex Settings →

Installing Additional Components

Several components of PE can be installed when needed instead of up-front.

Note that you cannot currently change which machines have the console and puppet master roles.

Installing Cloud Provisioner

You can install PE's cloud provisioning tools on any node by running the `puppet-enterprise-upgrader` script included in the installation tarball. This script will give you a chance to install the cloud provisioning tools. Using the upgrader script for the version of PE that is already installed should have no ill effect on the node's configuration.

Installing the Ruby Development Libraries

Puppet Enterprise ships its own copy of Ruby. If you need to install the Ruby development libraries, the installer tarball includes packages for them, which can be installed manually.

The installation method will depend on your operating system's package management tools, but in each case, you must first navigate to the directory containing the packages for your operating system and architecture, which will be inside the `packages` subdirectory of the unarchived Puppet Enterprise tarball.

For systems using apt and dpkg (Ubuntu and Debian), run the following commands:

```
$ sudo dpkg -i *ruby*dev*

$ sudo apt-get install --fix-broken
```

For systems using rpm and yum (Red Hat Enterprise Linux, CentOS, and Oracle Linux), run the following commands:

```
$ sudo yum localinstall --nogpgcheck *ruby*dev*
```

← Maintenance: Console Maintenance Tasks — Index — Maintenance: Reconfiguring Complex Settings →

PE 2.0 » Maintenance and Troubleshooting » Reconfiguring PE

Reconfiguring Complex Settings

During installation, you make several major decisions that affect the way Puppet Enterprise works. Several of these decisions can be changed after the fact.

Changing the Console's Port

By default, a new installation of PE will serve the console on port 443. However, previous versions of PE served the console's predecessor on port 3000. If you upgraded and want to change to the more convenient new default, or if you need port 443 for something else and want to shift the console somewhere else, perform the following steps:

- Stop the `pe-httpd` service:

```
$ sudo /etc/init.d/pe-httpd stop
```

- Edit `/etc/puppetlabs/httpd/conf.d/puppetdashboard.conf` on the console server, and change the port number in the `Listen 443` and `<VirtualHost *:443>` directives. (These directives will contain the current port, which is not necessarily 443.)
- Edit `/etc/puppetlabs/puppet/puppet.conf` on the puppet master server, and change the `reporturl` setting to use your preferred port.
- Edit `/etc/puppetlabs/puppet-dashboard/external_node` on the puppet master server, and change the `ENC_BASE_URL` to use your preferred port.
- Make sure to allow access to the new port in your system's firewall rules.
- Start the `pe-httpd` service:

```
$ sudo /etc/init.d/pe-httpd start
```

Changing the Console's User/Password

Access to the console is controlled with a user name and password, which are chosen during installation. You can change the password or add a new user using PE's copy of `htpasswd` on the `/etc/puppetlabs/httpd/console_passwd` file. To change the password of the default "console" user:

```
$ sudo /opt/puppet/bin/htpasswd /etc/puppetlabs/httpd/console_passwd console
New password:
Re-type new password:
Updating password for user console
```

To add a new user:

```
$ sudo /opt/puppet/bin/htpasswd /etc/puppetlabs/httpd/console_passwd new_admin
```

Run `/opt/puppet/bin/htpasswd --help` for more info, including how to delete users.

Note that you do not need to change any other Puppet settings after changing this password; it is only used for the console's web interface.

Changing the Console's Database User/Password

The console uses a database user account to access its MySQL database. If this user's password is compromised, or if it just needs to be changed periodically for policy reasons, do the following:

1. Stop the `pe-httpd` service on the console server:

   ```
   $ sudo /etc/init.d/pe-httpd stop
   ```

2. Use the MySQL administration tool of your choice to change the user's password. With the standard `mysql` client, you can do this with:

   ```
   SET PASSWORD FOR 'console'@'localhost' = PASSWORD('<new password>');
   ```

3. Edit `/etc/puppetlabs/puppet-dashboard/database.yml` on the console server and change the `password:` line under "common" (or under "production," depending on your configuration) to contain the new password.
4. Edit `/etc/puppetlabs/puppet/puppet.conf` on the console server and change the `dbpassword =` line (under `[master]`) to contain the new password.
5. Start the `pe-httpd` service back up:

   ```
   $ sudo /etc/init.d/pe-httpd start
   ```

Changing Orchestration Security

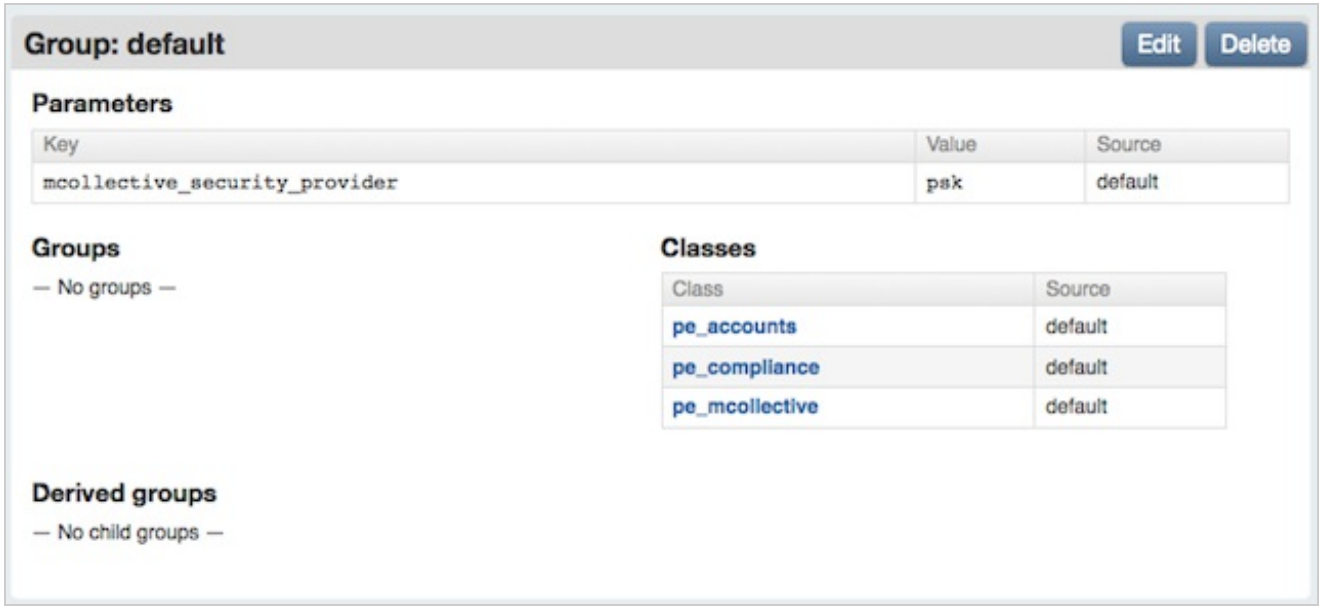
PE's orchestration messages are encrypted and authenticated. You can easily change the authentication method or the password used in the PSK authentication method.

Changing the Authentication Method

By default, orchestration messages are encrypted with SSL and authenticated with a pre-shared key (PSK). This balance of security and performance should work for most users, but if you need higher security and don't have a large number of nodes, you can reconfigure PE to authenticate orchestration messages with an AES key pair.

You can change the authentication method via the console. Navigate to your default group, then create a new parameter called `mcollective_security_provider`. The value of this parameter can be:

- `psk` — Use the default PSK authentication.
- `aespe_security` — Use AES authentication for extra security.



After changing the authentication method, you cannot issue orchestration messages to a given node until it has run Puppet at least once. This means changing the authentication method requires a 30 minute maintenance window during which orchestration will not be used. You can check whether a given node has changed its orchestration settings by checking its recent reports in the console and ensuring that its `/etc/puppetlabs/mcollective/server.cfg` file was modified.□

Before changing the authentication method, you should carefully consider the pros and cons of each:

PSK

The PSK authentication method is enabled by default. Under this method, nodes receive a secret key via Puppet and trust messages sent by clients who have the same key.

Pro:

- Scales to many hundreds of nodes on average puppet master hardware.

Con:

- Private key is known to all nodes — an attacker with elevated privileges on one node could obtain the pre-shared key and issue valid orchestration commands to other nodes.
- No protection from replay attacks — an attacker could repeatedly re-send commands without knowing their content.

AES (`AESPE_SECURITY`)

The AES authentication method must be manually enabled in the console. Under this method, nodes receive one or more public keys, and trust messages sent by clients who have one of the matching private keys.

Pro:

- Private key is only known to the puppet master node — an attacker with elevated privileges on an agent node cannot command other nodes.
- Protection from replay attacks — once a short time window has passed, messages cannot be re-sent, and must be reconstructed and encrypted with a private key.

Con:

- All nodes must have very accurate timekeeping, and their clocks must be in sync. (The allowable time variance defaults to a 60-second window.)
- The puppet master node requires more powerful hardware. This authentication method may not reliably scale to multiple hundreds of nodes.

Changing the Pre-Shared Key

When using PSK authentication, orchestration messages are authenticated with a randomly generated password. (This password is distributed to your nodes by Puppet, and isn't meant to be entered by users.)

If this password is compromised, or if it just needs to be changed periodically for policy reasons, you can do so by editing `/etc/puppetlabs/mcollective/credentials` on the puppet master server and then waiting for puppet agent to run on every node.

After changing the password, you cannot issue orchestration messages to a given node until it has run Puppet at least once. This means changing the orchestration password requires a 30 minute maintenance window during which orchestration will not be used. You can check whether a given node has changed its orchestration settings by checking its recent reports in the console and ensuring that its `/etc/puppetlabs/mcollective/server.cfg` file was modified.□

← Maintenance: Installing Additional Components — Index
