Puppet Enterprise 2.7 User's Guide

(Generated on November 16, 2012, from git revision fed21940866615ff2ef0f2c8ed6b3159932ba595)

Puppet Enterprise User's Guide

Welcome! This is the user's guide for Puppet Enterprise 2.7.

- If you are new to Puppet Enterprise, begin with the quick start guide to create a small proof-of-concept deployment and experience the core Puppet Enterprise tools and workflows. This guided walkthrough will take approximately 30 minutes.
- To install Puppet Enterprise, see the following pages:
  - System Requirements
  - Installing PE
  - Installing Windows Agents
  - Puppet Enterprise Downloads
- To see what's new since the last release, see New Features.

Otherwise, use the navigation to the left to move between this guide's sections and chapters.

About Puppet Enterprise

Thank you for choosing Puppet Enterprise, IT automation software that allows system administrators to programmatically provision, configure, and manage servers, network devices and storage, in the data center or in the cloud.

Puppet Enterprise (PE) offers:

- Configuration management tools that let sysadmins define a desired state for their infrastructure and then automatically enforce that state.
- A web-based console UI, for analyzing reports, managing your Puppet systems and users, and editing resources on the fly.
- Powerful orchestration capabilities.
- An alternate compliance workflow for auditing changes to unmanaged resources.
- Cloud provisioning tools for creating and configuring new VM instances.

This user's guide will help you start using Puppet Enterprise 2.7, and will serve as a reference as you gain more experience. It covers PE-specific features and offers brief introductions to Puppet and MCollective. Use the navigation at left to move between the guide's sections and chapters.

For New Users

If you've never used Puppet before and want to evaluate Puppet Enterprise, follow the Puppet Enterprise quick start guide. This walkthrough will guide you through creating a small proof-of-concept deployment while demonstrating the core features and workflows of Puppet Enterprise.

For Returning Users

See the what's new page for the new features in this release of Puppet Enterprise. You can find detailed release notes for updates within the 2.7.x series in the appendix of this guide.

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 configuration. By default, Puppet Enterprise uses a client/server Puppet deployment, where agent nodes fetch configurations from a central puppet master.

About Orchestration

Puppet Enterprise includes 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.

 NOTE: Orchestration and live management are not yet supported on Windows nodes.

About the Console

PE'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 Provisioning Tools

PE includes command line tools for building new nodes, which can create new VMware, Openstack

and Amazon EC2 instances, install PE on any virtual or physical machine, and classify newly provisioned nodes within your Puppet infrastructure.

Licensing

PE can be evaluated with a complimentary 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.

- Next: New Features

New Features in PE 2.7

Version 2.7.0

PE 2.7.0 adds new node request management capabilities to the console. It also fixes several bugs and adds to or modifies some existing capabilities.

NODE REQUEST MANAGEMENT

You can now use the console to view, approve or reject requests coming from nodes or services attempting to join your site. PE's new request management capabilities let you view incoming node requests, including their CSR's fingerprint, and accept or reject them singly or as a batch. For more information, see the Node Request Management page.

NEW AND UPDATED MODULES

Three new modules have been added in PE 2.7: `puppetlabs-certificate_manager`, `puppetlabs-auth_conf`, and `ripienaar-concat`. The new certificate_manager module, which provides the new node request management capabilities, uses the auth_conf module, which depends on the concat module, to update and manage `auth.conf` on the master and console.

In addition, `puppetlabs-stdlib` has been updated to version 2.5.1 to include new PE version facts puppetlabs-pe_compliance. For details, see Issue 16485.

PUPPET PATCHES

PE 2.7.0 uses a specially patched version of puppet 2.7.19. See the Appendix for release notes detailing these patches.

- Next: Getting Support

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

- Next: Quick Start

Quick Start: Using PE 2.7

Welcome to the PE 2.7 quick start guide. This document is a short walkthrough to help you evaluate Puppet Enterprise and become familiar with its features. Follow along to learn how to:

- Install a small proof-of-concept deployment
- Add nodes to your deployment
- Examine nodes in real time with live management
- Install a third-party Puppet module
- Apply Puppet classes to nodes with the console

> Following this walkthrough will take approximately 30 minutes.

Creating a Deployment

A standard Puppet Enterprise deployment consists of:

- Many agent nodes, which are computers managed by Puppet.
- At least one puppet master server, which serves configurations to agent nodes.
- At least one console server, which analyzes agent reports and presents a GUI for managing your site.

For this deployment, the puppet master and the console will be the same machine, and we will have one additional agent node.

> **Preparing Your Proof-of-Concept Systems**
>
> To create this small deployment, you will need the following:
>
> - At least two computers (“nodes”) running a *nix operating system supported by Puppet Enterprise.

    - These can be virtual machines or physical servers.
    - One of these nodes (the puppet master server) should have at least 1 GB of RAM. Note: For actual production use, a puppet master node should have at least 4 GB of RAM.

- Optionally, a computer running a version of Microsoft Windows supported by Puppet Enterprise.
- Puppet Enterprise installer tarballs suitable for the OS and architecture your nodes are using.
- A network — all of your nodes should be able to reach each other.
- An internet connection or a local mirror of your operating system's package repositories, for downloading additional software that Puppet Enterprise may require.
- Properly configured firewalls.
    - For demonstration purposes, all nodes should allow all traffic on ports 8140, 61613, and 443. (Production deployments can and should partially restrict this traffic.)

- Properly configured name resolution.
    - Each node needs a unique hostname, and they should be on a shared domain. For the rest of this walkthrough, we will refer to the puppet master as `master.example.com`, the first agent node as `agent1.example.com`, and the Windows node as `windows.example.com`. You can use any hostnames and any domain; simply substitute the names as needed throughout this document.
    - All nodes must know their own hostnames. This can be done by properly configuring reverse DNS on your local DNS server, or by setting the hostname explicitly. Setting the hostname usually involves the hostname command and one or more configuration files, while the exact method varies by platform.
    - All nodes must be able to reach each other by name. This can be done with a local DNS server, or by editing the `/etc/hosts` file on each node to point to the proper IP addresses. Test this by running `ping master.example.com` and `ping agent1.example.com` on every node, including the Windows node if present.
    - Optionally, to simplify configuration later, all nodes should also be able to reach the puppet master node at the hostname `puppet`. This can be done with DNS or with hosts files. Test this by running `ping puppet` on every node.
    - The control workstation from which you are carrying out these instructions must be able to reach every node in the deployment by name.

Installing the Puppet Master

- On the puppet master node, log in as root or with a root shell. (Use `sudo -s` to get a root shell if your operating system's root account is disabled, as on Debian and Ubuntu.)
- Download the Puppet Enterprise tarball, extract it, and navigate to the directory it creates.
- Run `./puppet-enterprise-installer`. The installer will ask a series of questions about which components to install, and how to configure them.

- Install the puppet master and console roles; the cloud provisioner role is not required, but may be useful if you later promote this machine to production.
- Make sure that the unique “certname” matches the hostname you chose for this node. (For example, `master.example.com`.)
- You will need the email address and console password it requests in order to use the console; choose something memorable.
- None of the other passwords are relevant to this quick start guide. Choose something random.
- Accept the default responses for every other question by hitting enter.

- The installer will then install and configure Puppet Enterprise. It will probably need to install additional packages from your OS’s repository, including Java and MySQL.

> You have now installed the puppet master node. A puppet master node is also an agent node, and can configure itself the same way it configures the other nodes in a deployment. Stay logged in as root for further exercises.

Installing the Agent Node

- On the agent node, log in as root or with a root shell. (Use `sudo -s` to get a root shell if your operating system’s root account is disabled.)
- Download the Puppet Enterprise tarball, extract it, and navigate to the directory it creates.
- Run `./puppet-enterprise-installer`. The installer will ask a series of questions about which components to install, and how to configure them.
  - Skip the puppet master and console roles; install the puppet agent role. The cloud provisioner role is optional and is not used in this exercise.
  - Make sure that the unique “certname” matches the hostname you chose for this node. (For example, `agent1.example.com`.)
  - Set the puppet master hostname as `master.example.com`. If you configured the master to be reachable at `puppet`, you can alternately accept the default.
  - Accept the default responses for every other question by hitting enter.

- The installer will then install and configure Puppet Enterprise.

> You have now installed the puppet agent node. Stay logged in as root for further exercises.

Installing the Optional Windows Node

- On the Windows node, log in as a user with administrator privileges.
- Download the Puppet Enterprise installer for Windows.
- Run the Windows installer by double-clicking it. The installer will ask for the name of the puppet

master to connect to; set this to `master.example.com`.

> You have now installed the Windows puppet agent node. Stay logged in as administrator for further exercises.

Adding Nodes to a Deployment

After installing, the agent nodes are not yet allowed to fetch configurations from the puppet master; they must be explicitly approved and granted a certificate.

Approving the Certificate Request

During installation, the agent node contacted the puppet master and requested a certificate. To add the agent node to the deployment, approve its request on the puppet master.

- On the puppet master node, run `puppet cert list` to view all outstanding certificate requests.
- Note that nodes called `agent1.example.com` and `windows.example.com` (or whichever names you chose) have requested certificates, and fingerprints for the requests are shown.
- On the puppet master node, run `puppet cert sign agent1.example.com` to approve the request and add the node to the deployment. Run `puppet cert sign windows.example.com` to approve the Windows node.

> The agent nodes can now retrieve configurations from the master.

Testing the Agent Nodes

During this walkthrough, we will be running puppet agent interactively. Normally, puppet agent runs in the background and fetches configurations from the puppet master every 30 minutes. (This interval is configurable with the `runinterval` setting in puppet.conf.)

- On the first agent node, run `puppet agent --test`. This will trigger a single puppet agent run with verbose logging.
- Note the long string of log messages, which should end with `notice: Finished catalog run in [...] seconds`.
- On the Windows node, open the start menu, navigate to the Puppet Enterprise folder, and choose “Run Puppet Agent,” elevating privileges if necessary.
- Note the similar string of log messages.

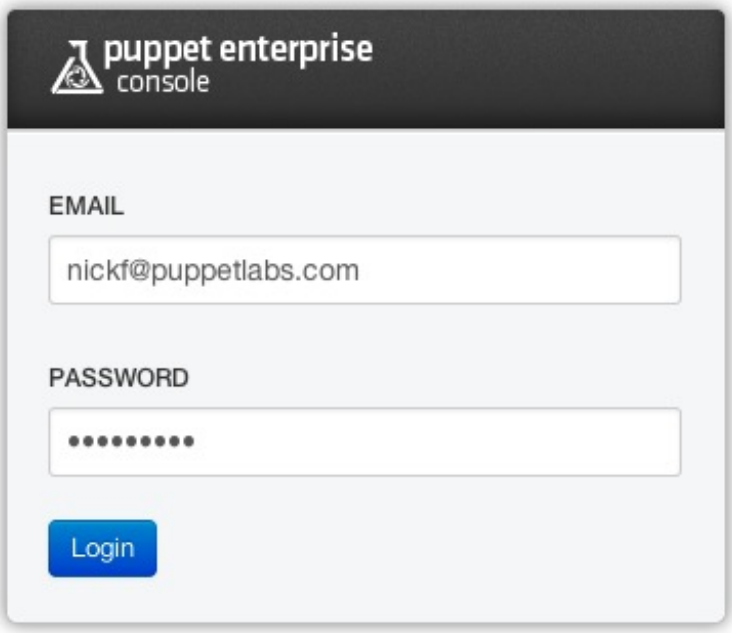
> You are now fully managing these nodes. They have checked in with the puppet master for the first time, and will continue to check in and fetch new configurations every 30 minutes. They will also appear in the console, where you can make changes to them by assigning

classes.

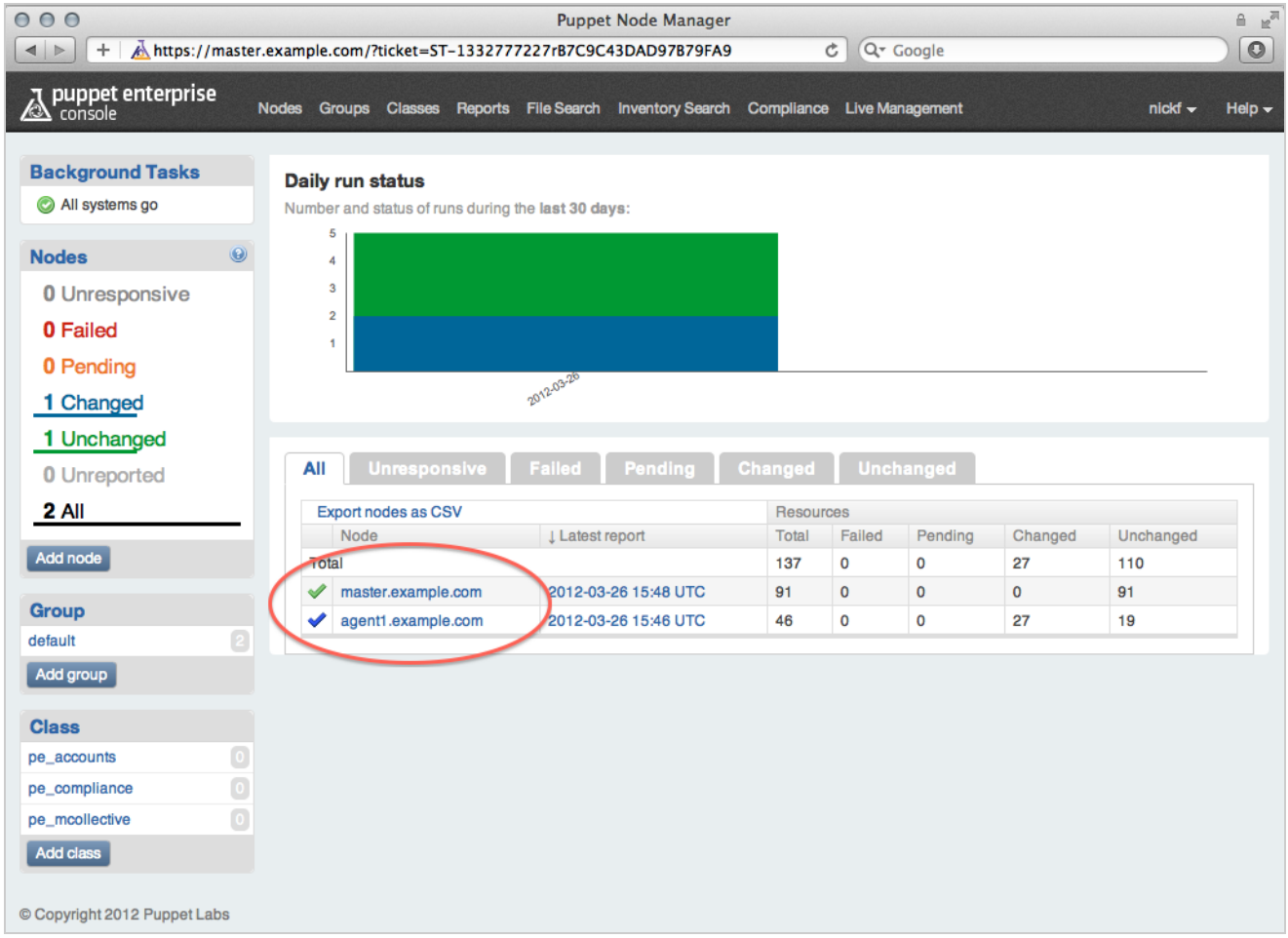
Viewing the Agent Nodes in the Console

You can now log into the console and see all agent nodes, including the puppet master node.

- On your control workstation, open a web browser and point it to https://master.example.com.
- You will receive a warning about an untrusted certificate. This is because you were the signing authority for the console's certificate, and your Puppet Enterprise deployment is not known to the major browser vendors as a valid signing authority. Ignore the warning and accept the certificate. The steps to do this vary by browser; see here for detailed steps for the major web browsers.
- Next, you will see a login screen for the console. Log in with the email address and password you provided when installing the puppet master.



- Next, you will see the front page of the console, which shows a summary of your deployment's recent puppet runs. Notice that the master and any agent nodes appear in the list of nodes:



- Explore the console. Note that if you click on a node to view its details, you can see its recent history, the Puppet classes it receives, and a very large list of inventory information about it. See here for more information about navigating the console.

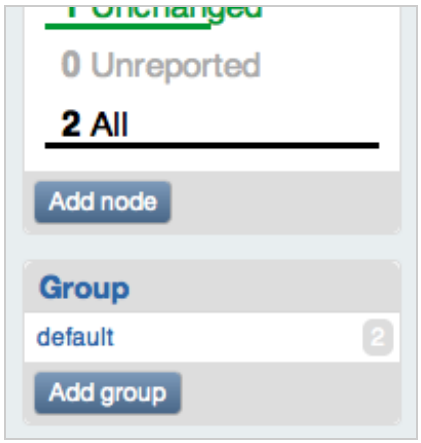
You now know how to find detailed information about any node in your deployment, including its status, inventory details, and the results of its last Puppet run.

Avoiding the Wait

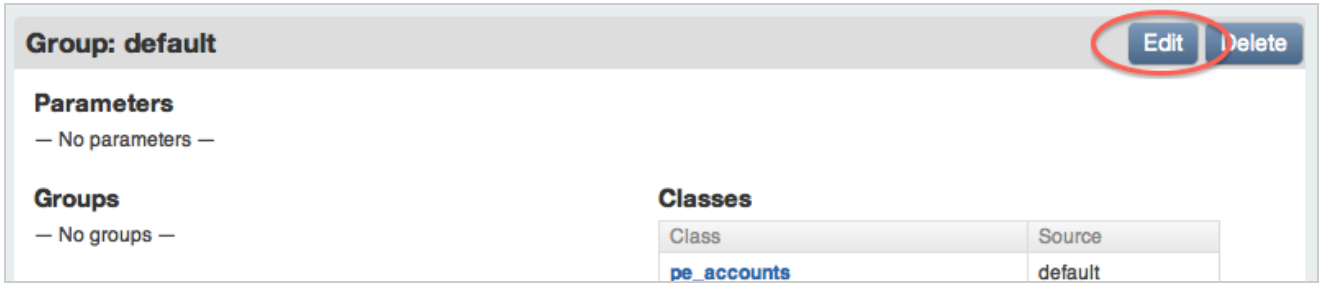
Although puppet agent is now fully functional on any agent nodes, some other Puppet Enterprise software is not; specifically, the daemon that listens for orchestration messages is not configured. This is because Puppet Enterprise uses Puppet to configure itself.

Puppet Enterprise does this automatically within 30 minutes of a node's first check-in. To fast-track the process and avoid the wait, do the following:

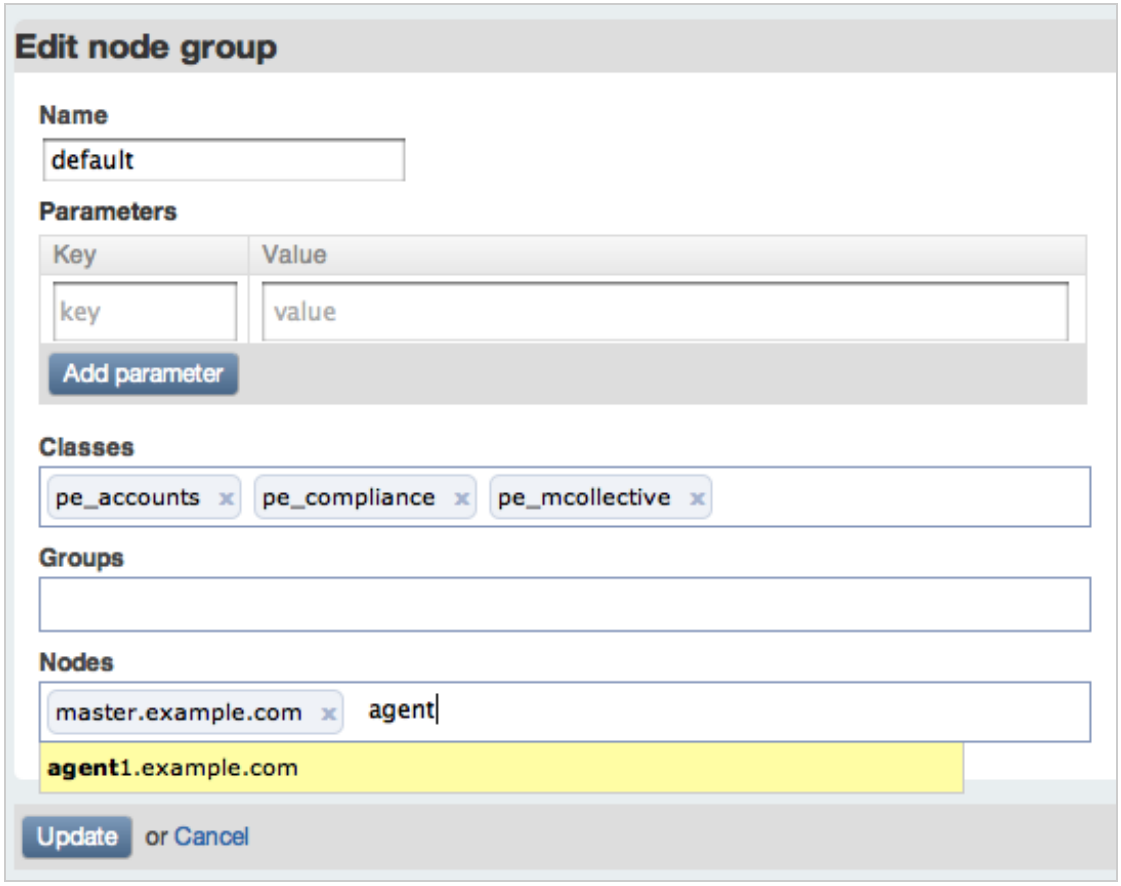
- On the console, use the sidebar to navigate to the default group:



- Check the list of nodes at the bottom of the page for `agent1.example.com` — depending on your timing, it may already be present. If so, skip the next two steps and go directly to the agent node.
- If `agent1` is not a member of the group already, click the “edit” button:



- In the “nodes” field, begin typing `agent1.example.com`’s name. You can then select it from the list of autocompletion guesses. Click the update button after you have selected it.



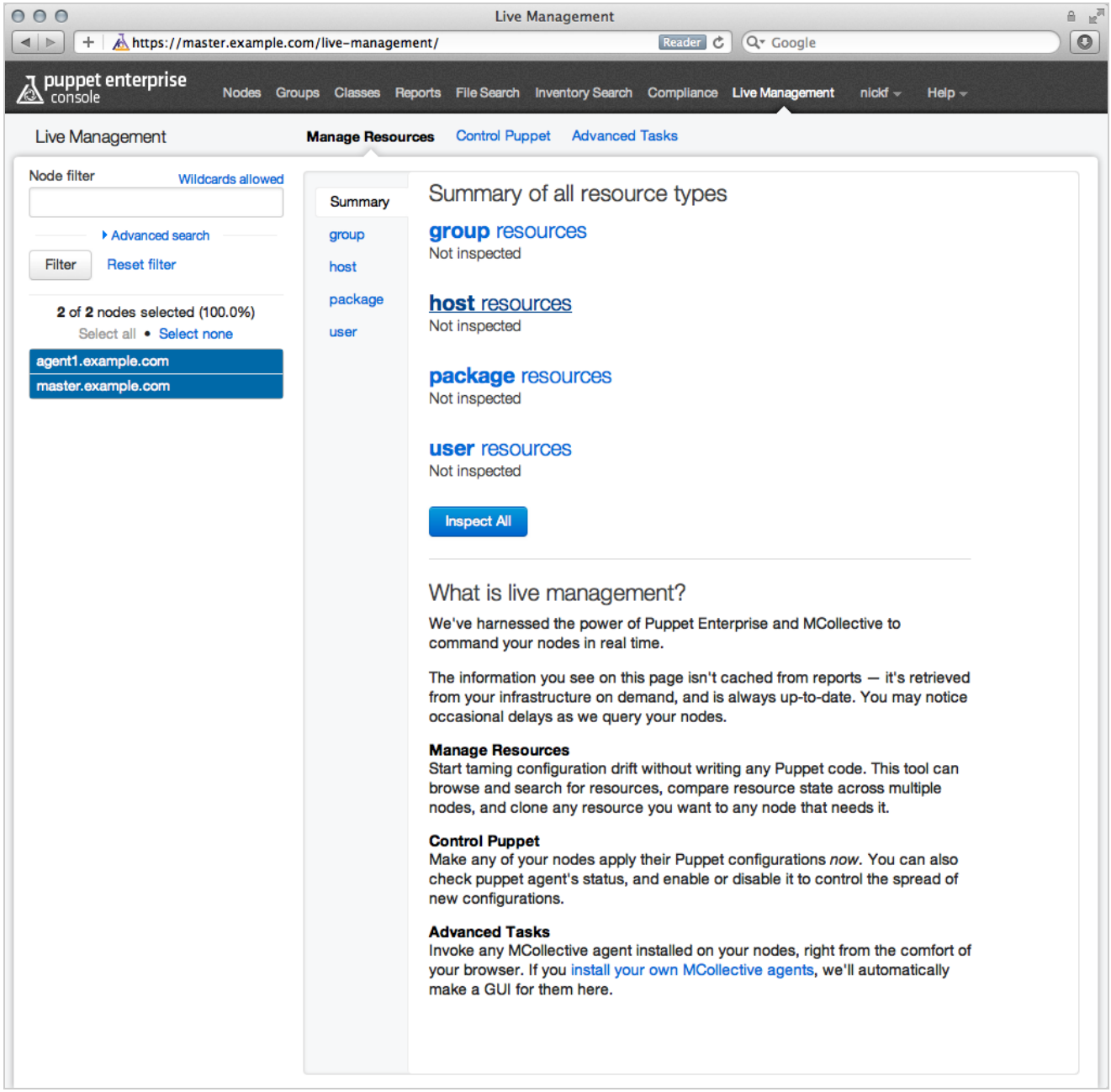
- On the first agent node, run `puppet agent --test` again. Note the long string of log messages related to the `pe_mcollective` class.
- You do not need to repeat this for the Windows node. Orchestration is not supported on Windows for this release of Puppet Enterprise.

> The first agent node can now respond to orchestration messages, and its resources can be edited live in the console.

Using Live Management to Control Agent Nodes

Live management uses Puppet Enterprise's orchestration features to view and edit resources in real time. It can also trigger Puppet runs and orchestration tasks.

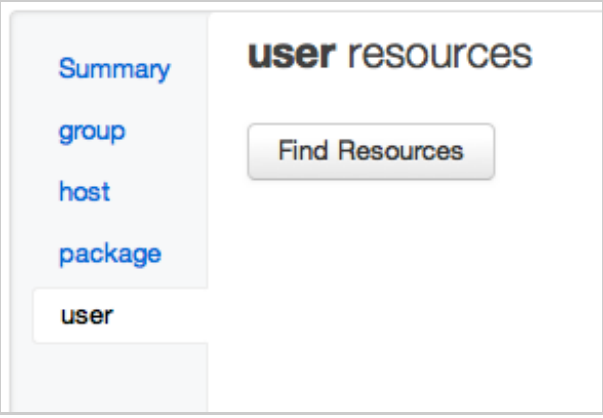
- On the console, click the "Live Management" tab in the top navigation.



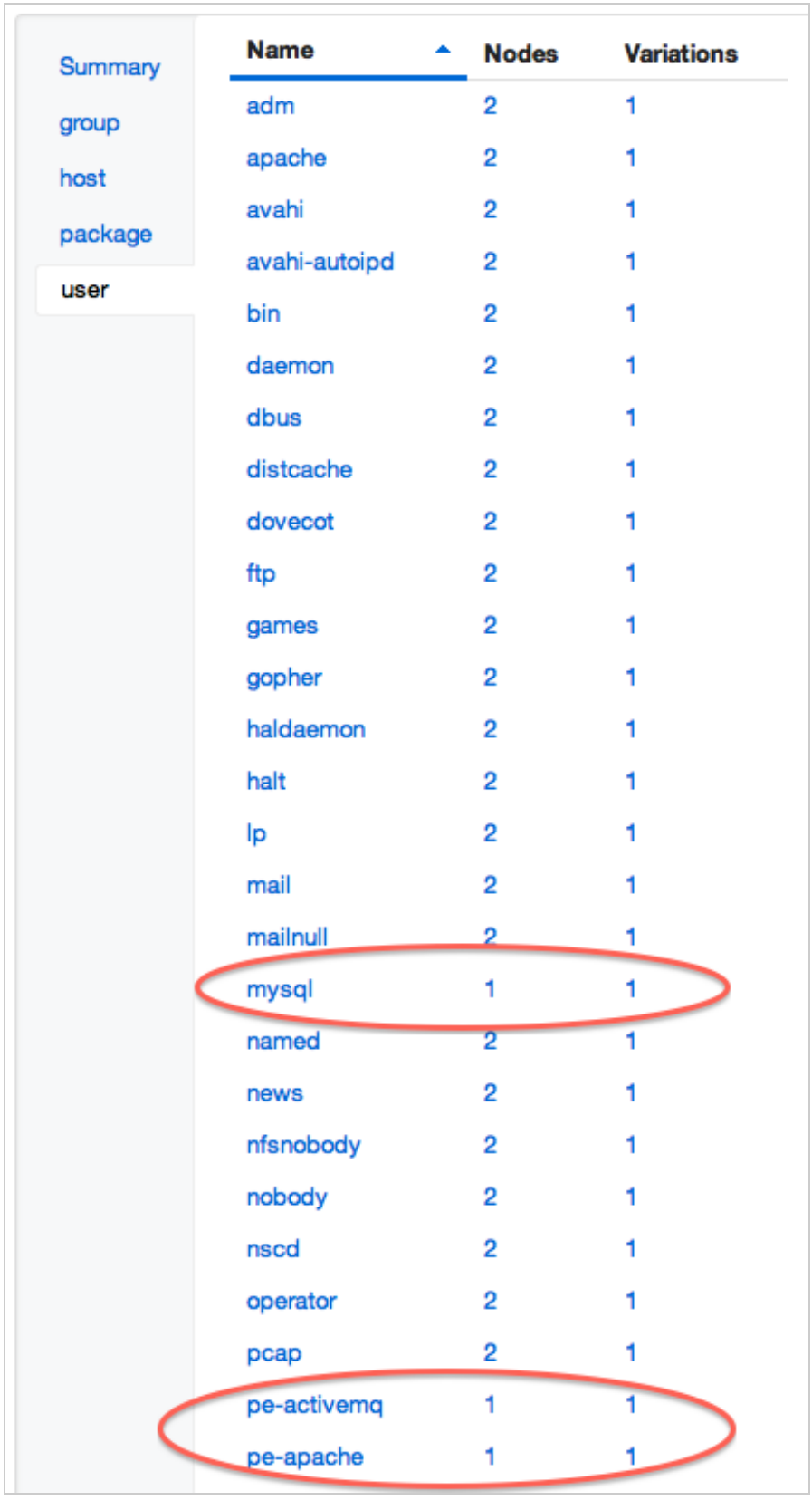
- Note that the master and the first agent node are visible in the sidebar, but the Windows node is not. Live management is not supported on Windows nodes for this release of Puppet Enterprise.

Discovering and Cloning Resources

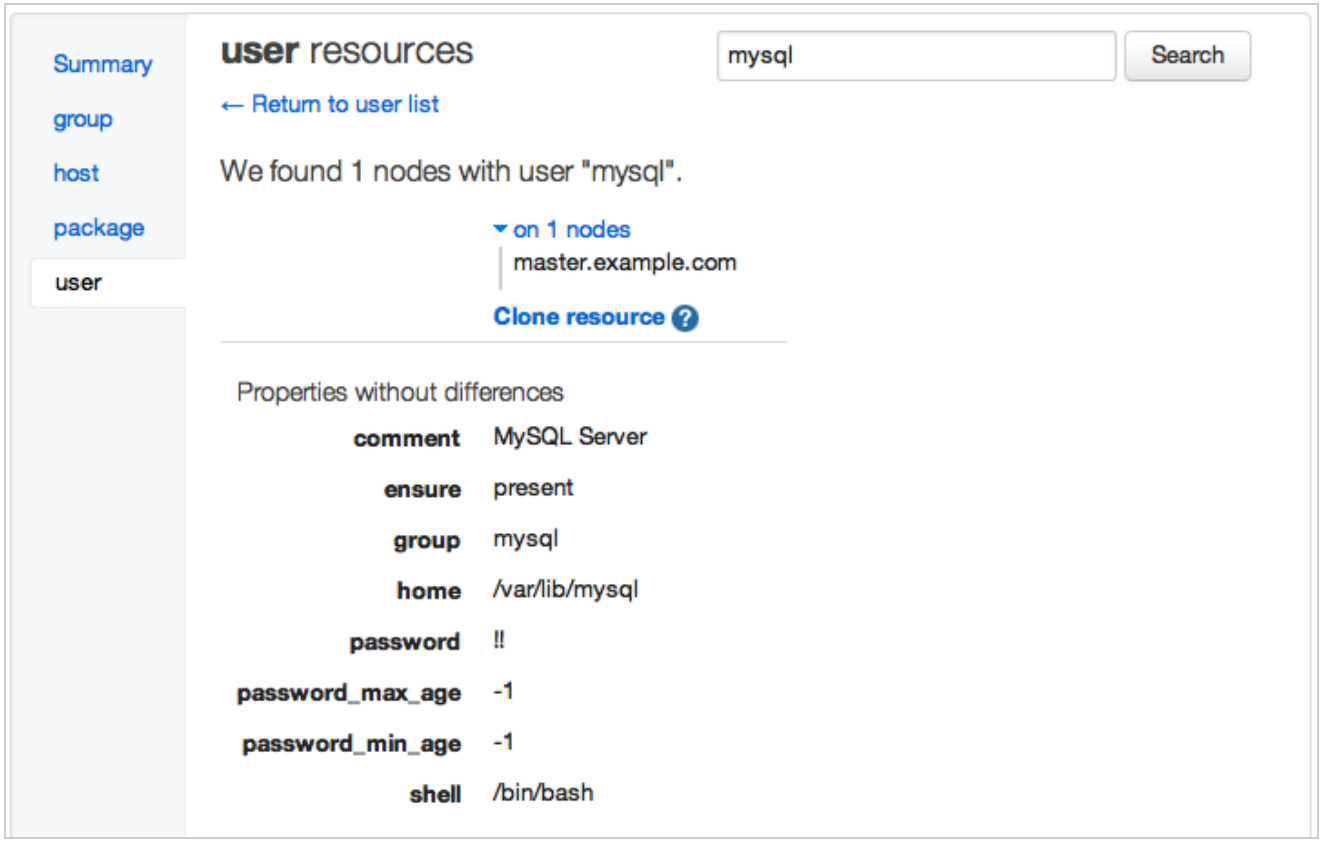
- Note that you are currently in the “manage resources” tab. Click the “user resources” link, then click the “find resources” button:



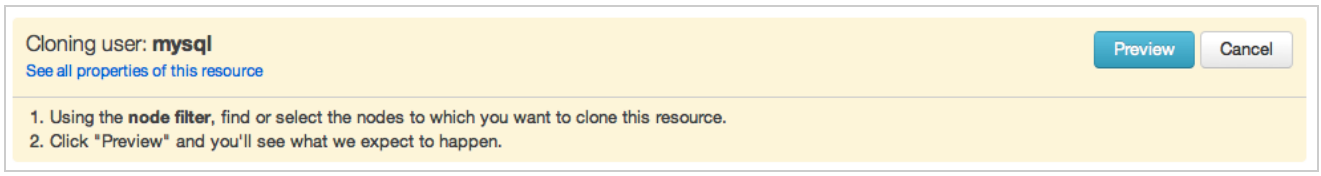
- Examine the complete list of user accounts found on all of the nodes currently selected in the sidebar node list. (In this case, both the master and the first agent node are selected.) Most of the users will be identical, as these machines are very close to a default OS install, but some users related to the puppet master's functionality are only on one node:

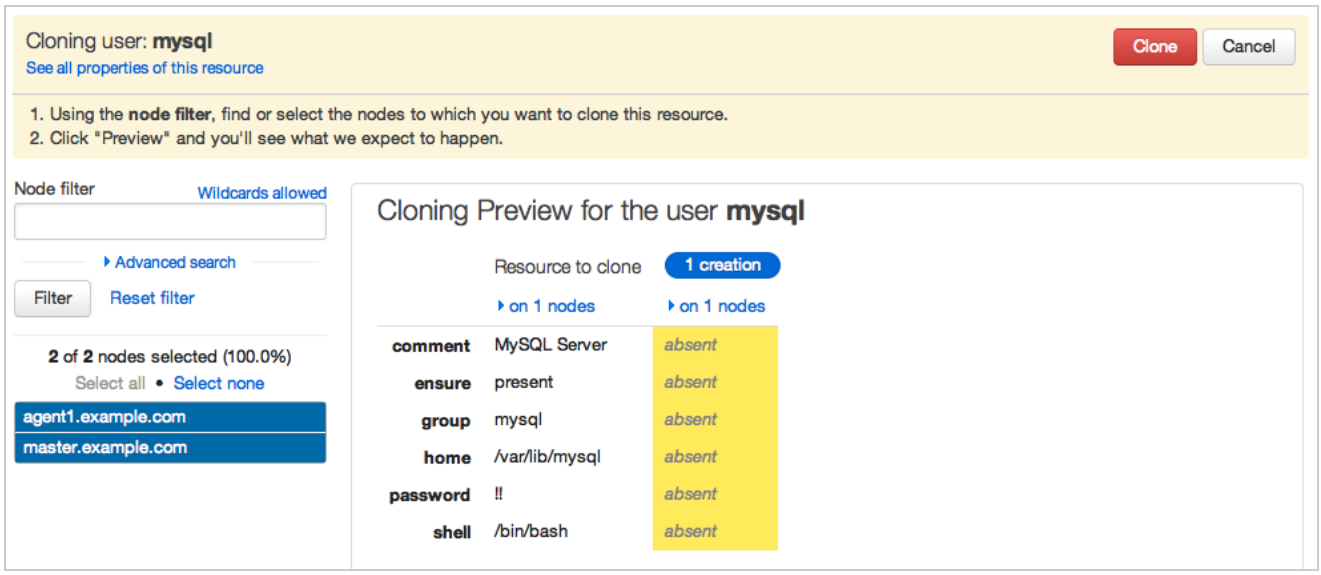


- Click the MySQL user, which is only present on the puppet master. (If the MySQL server was installed on both nodes, you can use a different user like `peadmin`.)

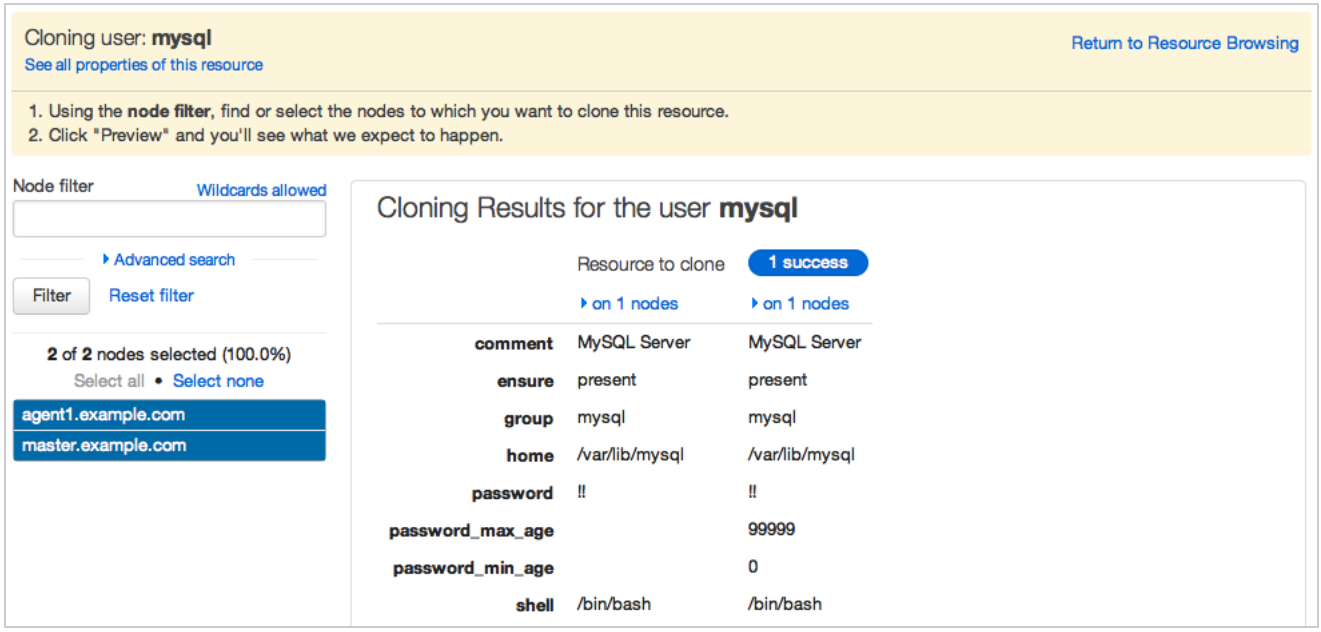


- Click the “Clone resource” link, then click the blue “Preview” button that appears. This will prepare the console to duplicate the mysql user across all of the nodes selected in the sidebar.





- Click the red “Clone” button to finish.□

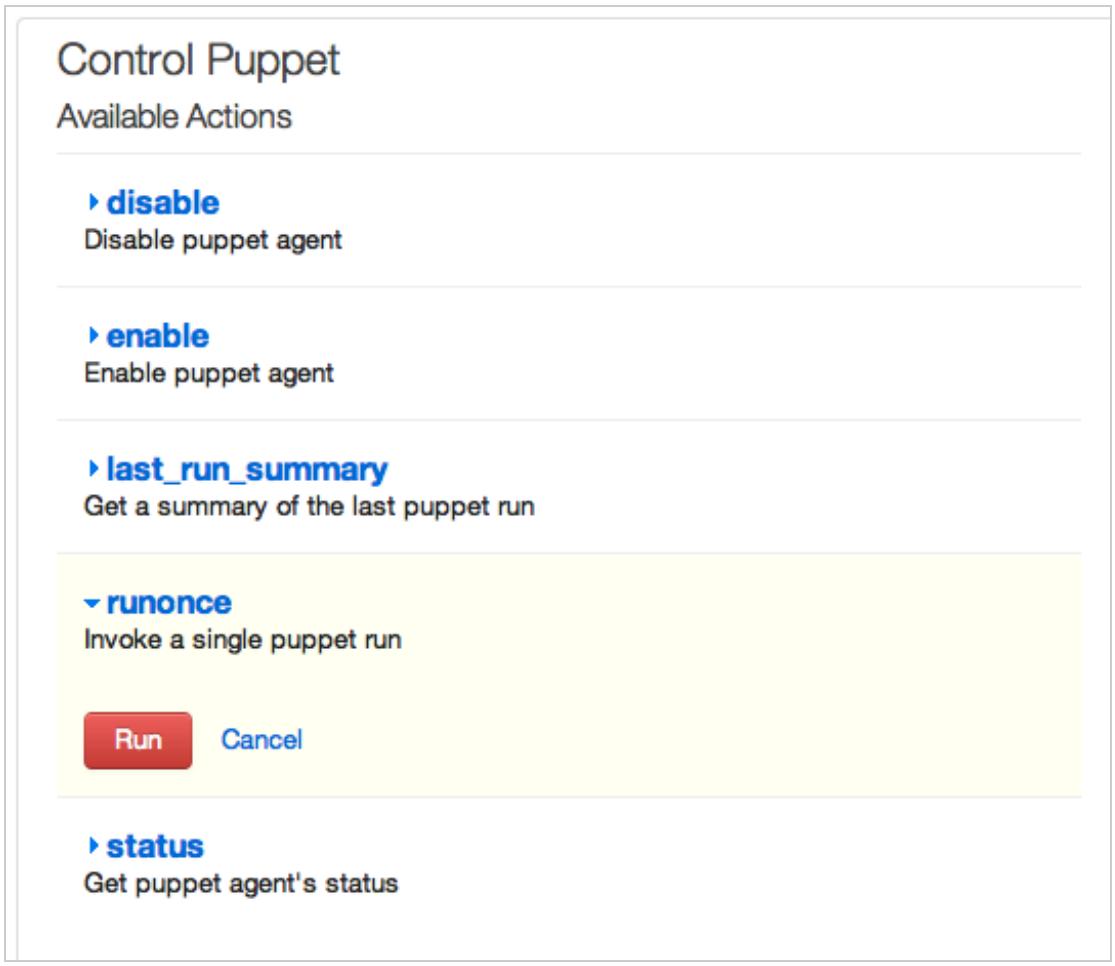


The `mysql` user is now present on both the master and the first agent node.

You can clone user accounts, user groups, entries in the `/etc/hosts` file, and software packages using this interface. This can let you quickly make many nodes resemble a single model node.

Triggering Puppet Runs

- On the console, in the live management page, click the “control puppet” tab.
- Click the “runonce” action to reveal the red “Run” button, then click the “Run” button.



You have just triggered a puppet agent run on several agents at once; in this case, the master and the first agent node. The “runonce” action will trigger a puppet run on every node currently selected in the sidebar.

In production deployments, select target nodes carefully, as running this action on dozens or hundreds of nodes at once can put strain on the puppet master server.

Installing a Puppet Module

Puppet configures nodes by applying classes to them. Classes are chunks of Puppet code that configure a specific aspect or feature of a machine.

Puppet classes are distributed in the form of modules. You can save time by using pre-existing modules. Pre-existing modules are distributed on the Puppet Forge, and can be installed with the `puppet module` subcommand. Any module installed on the puppet master can be used to configure agent nodes.

Installing two Forge Modules

- On your control workstation, navigate to http://forge.puppetlabs.com/puppetlabs/motd. This is the Forge listing for an example module that sets the message of the day file (`/etc/motd`), which is displayed to users when they log into a *nix system.

- Navigate to https://forge.puppetlabs.com/puppetlabs/win_desktop_shortcut. This is the Forge listing for an example module that manages a desktop shortcut on Windows.
- On the puppet master, run `puppet module search motd`. This is an alternate way to find the same information as a Forge listing contains:

```
  Searching http://forge.puppetlabs.com ...
  NAME             DESCRIPTION
AUTHOR        KEYWORDS
  puppetlabs-motd  This module populates `/etc/motd` with the contents of ...
@puppetlabs   Testing
  jeffmccune-motd  This manages a basic message of the day based on useful...
@jeffmccune   motd
  dhoppe-motd       This module manages motd
@dhoppe       debian ubuntu motd
  saz-motd         Manage 'Message Of The Day' via Puppet
@saz          motd
```

- Install the first module by running `puppet module install puppetlabs-motd`:

```
  Preparing to install into /etc/puppetlabs/puppet/modules ...
  Downloading from http://forge.puppetlabs.com ...
  Installing -- do not interrupt ...
  /etc/puppetlabs/puppet/modules
  └── puppetlabs-motd (v1.0.0)
```

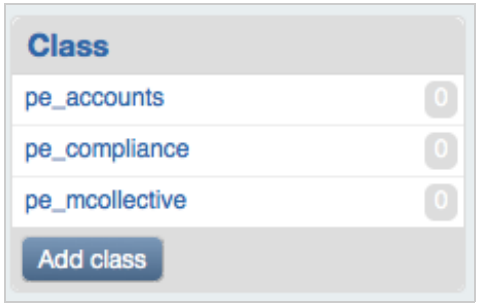
- Install the second module by running `puppet module install puppetlabs-win_desktop_shortcut`. (If you are not using any Windows nodes, this module is inert; you can install it or skip it.)

> You have just installed multiple Puppet modules. All of the classes in them are now available to be added to the console and assigned to nodes.

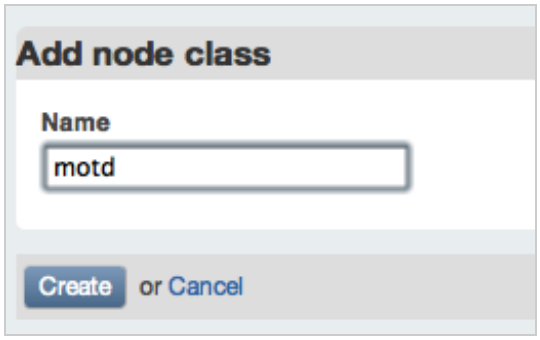
Using Modules in the Console

Every module contains one or more classes. The modules you just installed contain classes called `motd` and `win_desktop_shortcut`. To use any class, you must tell the console about it and then assign it to one or more nodes.

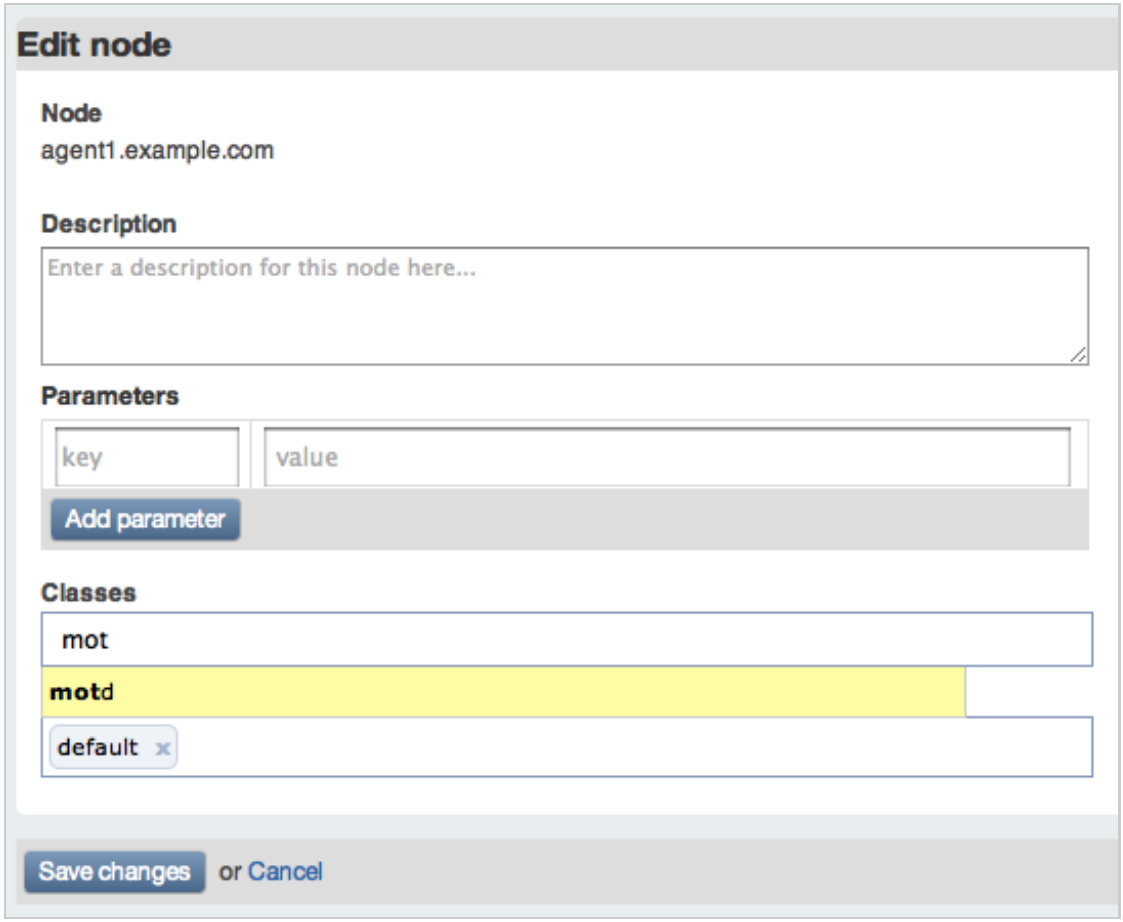
- On the console, click the “Classes” link in the top navigation bar, then click the “Add class” button in the sidebar:



- Type the name of the `motd` class, and click the “create” button:



- Do the same for the `win_desktop_shortcut` class.
- Navigate to `agent1.example.com` (by clicking the “Nodes” link in the top nav bar and clicking `agent1`’s name), click the “Edit” button, and begin typing “motd” in the “classes” field; you can select the `motd` class from the list of autocomplete suggestions. Click the “Save changes” button after you have selected it.



- Note that the `motd` class now appears in the list of `agent1`'s classes.
- Navigate to `windows.example.com`, click the edit button, and begin typing "`win_desktop_shortcut`" in the "classes" field; select the class and click the "save changes" button.
- Note that the `win_desktop_shortcut` class now appears in the list of `windows.example.com`'s classes.
- Navigate to the live management page, and select the "control Puppet" tab. Use the "runonce" action to trigger a puppet run on both the master and the first agent. Wait one or two minutes.
- On the first agent node, run `cat /etc/motd`. Note that its contents resemble the following:

```
The operating system is CentOS
The free memory is 82.27 MB
The domain is example.com
```

- On the puppet master, run `cat /etc/motd`. Note that its contents are either empty or the operating system's default, since the `motd` class wasn't applied to it.
- On the Windows node, choose "Run Puppet Agent" from the start menu, elevating privileges if necessary.
- View the desktop; note that there is a shortcut to the Puppet Labs website.

Puppet is now managing the first agent node's message of the day file, and will revert it to the specified state if it is ever modified. Puppet is also managing the desktop shortcut on the Windows machine, and will restore it if it is ever deleted or modified.

For more recommended modules, search the Forge or check out the Module of the Week series on the Puppet Labs blog.

Summary

You have now experienced the core features and workflows of Puppet Enterprise. In summary, a Puppet Enterprise user will:

- Deploy new nodes, install PE on them, and add them to their deployment by approving their certificate requests.
- Use pre-built modules from the Forge to save time and effort.
- Assign classes to nodes in the console.
- Use live management for ad-hoc edits to nodes, and for triggering puppet agent runs when necessary.

Next

In addition to what this walkthrough has covered, most users will also:

- Edit modules from the Forge to make them better suit the deployment.
- Create new modules from scratch by writing classes that manage resources.
- Examine reports in the console.
- Use a site module to compose other modules into machine roles, allowing console users to control policy instead of implementation.
- Assign classes to groups in the console instead of individual nodes.

To learn about these workflows, continue to the writing modules quick start guide.

To explore the rest of the PE 2.7 user's guide, use the sidebar at the top of this page, or return to the index.

- Next: Quick Start: Writing Modules

Quick Start: Writing Modules for PE 2.7

Welcome to the PE 2.7 advanced quick start guide. This document is a continuation of the quick start guide, and is a short walkthrough to help you become more familiar with PE's features. Follow along to learn how to:

- Modify modules obtained from the Forge

- Write your own Puppet module
- Examine reports in the console
- Create a site module that composes other modules into machine roles
- Apply Puppet classes to groups with the console

> Before starting this walkthrough, you should have completed the previous quick start guide. You should still be logged in as root or administrator on your nodes.

Editing a Forge Module

Although many Forge modules are exact solutions that fit your site, many more are almost what you need. Most users will edit many of their Forge modules.

Module Basics

Modules are stored in `/etc/puppetlabs/puppet/modules`. (This can be configured with the `modulepath` setting in `puppet.conf`.)

Modules are directories. Their basic structure looks like this:

- `motd/` (the module name)
  - `manifests/`
    - `init.pp` (contains the `motd` class)
    - `public.pp` (contains the `motd::public` class)

Every manifest (.pp) file contains a single 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>`.

Many modules contain directories other than `manifests`; these will not be covered in this guide.

- For more on how modules work, see Module Fundamentals in the Puppet documentation.
- For a more detailed guided tour, see the module chapters of Learning Puppet.

Editing a Manifest

This exercise will modify the desktop shortcut being managed on your Windows node.

- On the puppet master, navigate to the modules directory by running `cd /etc/puppetlabs/puppet/modules`.
- Run `ls` to view the currently installed modules; note that `motd` and `win_desktop_shortcut` are present.

- Open and begin editing `win_desktop_shortcut/manifests/init.pp`, using the text editor of your choice.
  - If you do not have a preferred Unix text editor, run `nano win_desktop_shortcut/manifests/init.pp`.
    - If Nano is not installed, run `puppet resource package nano ensure=installed` to install it from your OS's package repositories.
- Note that the desktop shortcut is being managed as a `file` resource, and its content is being set with the `content` attribute:

```
class win_desktop_shortcut {

  if $osfamily == "windows" {
    if $win_common_desktop_directory {

      file { "${win_common_desktop_directory}\\PuppetLabs.URL":
        ensure  => present,
        content => "[InternetShortcut]\nURL=http://puppetlabs.com",
      }

    }
  }

}
```

For more on resource declarations, see the manifests chapter of Learning Puppet or the resources section of the language guide. For more about how file paths with backslashes work in manifests for Windows, see the page on writing manifests for Windows.

- Change the `ensure` attribute of the `file` resource to `absent`.
- Delete the `content` line of the `file` resource.
- Create two new `file` resources to manage other files on the desktop, mimicking the structure of the first resource:

```
    file { "${win_common_desktop_directory}\\RunningPuppet.URL":
      ensure  => present,
      content =>
"[InternetShortcut]\nURL=http://docs.puppetlabs.com/windows/running.html",
    }

    file { "${win_common_desktop_directory}\\Readme.txt":
      ensure  => present,
      content => "This node is managed by Puppet. Some files and services
cannot be edited locally; contact your sysadmin for details.",
    }
```

Make sure that these resources are within the two “if” blocks, alongside the first resource.

- Save and close the file.
- On the Windows node, choose “Run Puppet Agent” from the Start menu, elevating privileges if necessary.
- Note that the original shortcut is gone, and a new shortcut and `Readme.txt` file have appeared on the desktop.

> Your copy of the Windows example module now behaves differently.
>
> If you had deleted the original resource instead of setting `ensure` to `absent`, it would have become a normal, unmanaged file — Puppet would not have deleted it, but it also would not have restored it if a local user were to delete it. Puppet does not care about resources that are not declared.

Editing a Template

- On the puppet master, navigate to the modules directory by running `cd /etc/puppetlabs/puppet/modules`.
- Open and begin editing `motd/manifests/init.pp`, using the text editor of your choice.
- Note that the content of the `motd` file is being filled with the `template` function, referring to a template within the module:

```
class motd {
  if $kernel == "Linux" {
    file { '/etc/motd':
      ensure  => file,
      backup  => false,
      content => template("motd/motd.erb"),
    }
  }
}
```

- Close the manifest file, then open and begin editing `motd/templates/motd.erb`.
- Add the line `Welcome to <%= hostname %>` at the beginning of the template file.
- Save and close the file.
- On the first agent node, run `puppet agent --test`, then log out and log back in.
- Note that the message of the day has changed to show the machine’s hostname.

> Your copy of the `motd` module now behaves differently.
>
> - For more about templates, see the templates chapter of Learning Puppet or the templates

section of the Puppet documentation.

- For more about variables, including “facts” like `hostname`, see the variables chapter of Learning Puppet or the variables section of the language guide.

Writing a Puppet Module

Third-party modules save time, but most users will also write their own modules.

Writing a Class in a Module

This exercise will create a class that manages the permissions of the `fstab`, `passwd`, and `crontab` files.

- Run `mkdir -p core_permissions/manifests` to create the module directory and manifests directory.
- Create and begin editing the `core_permissions/manifests/init.pp` file.
- Edit the init.pp file so it contains the following, then save it and exit the editor:

```
class core_permissions {
  if $osfamily != 'windows' {

    $rootgroup = $operatingsystem ? {
      'Solaris' => 'wheel',
      default   => 'root',
    }

    file {'/etc/fstab':
      ensure => present,
      mode   => 0644,
      owner  => 'root',
      group  => "$rootgroup",
    }

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

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

  }
}
```

You have created a new module containing a single class. Puppet now knows about this class, and it can be added to the console and assigned to nodes.

This new class:

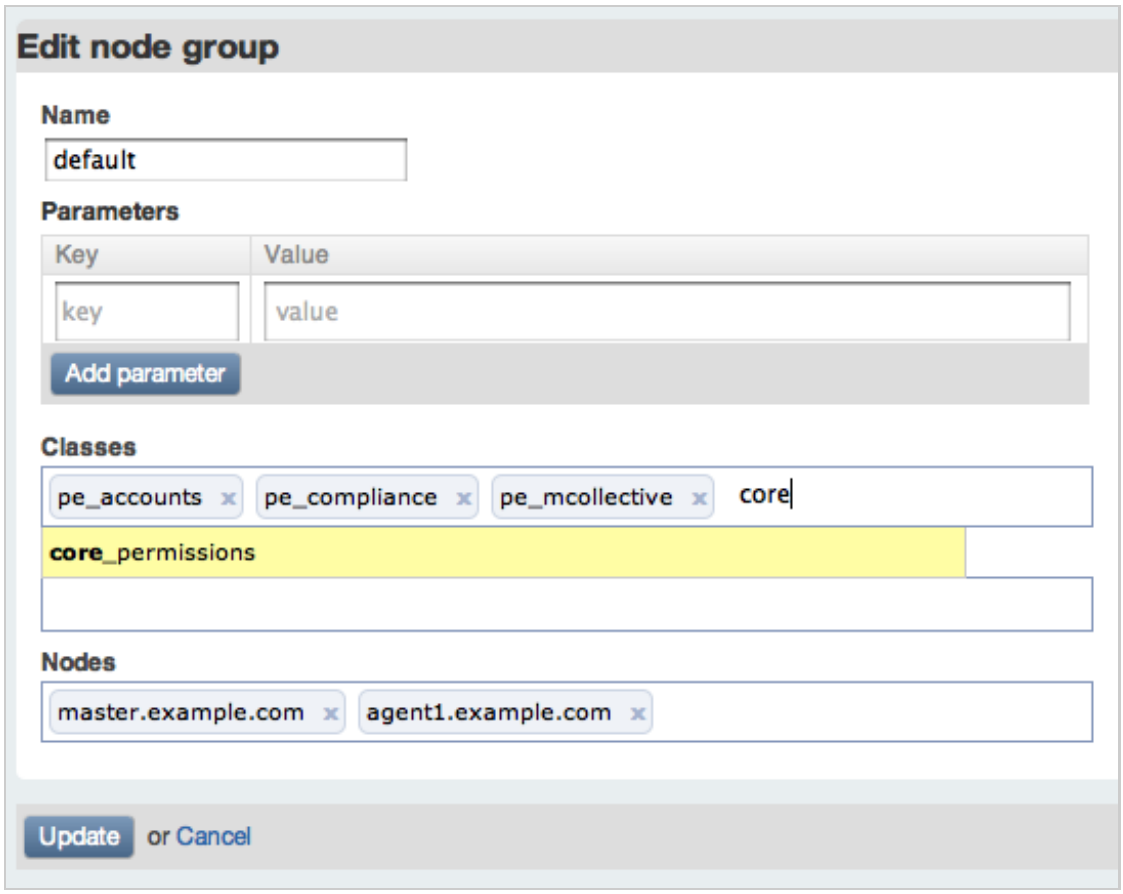
- Uses an "if" conditional to only manage *nix systems.
- Uses a selector conditional and a variable to change the name of the root user's primary group on Solaris.
- Uses three `file` resources to manage the `fstab`, `passwd`, and `crontab` files on *nix systems. These resources do not manage the content of the files, only their ownership and permissions.

See the Puppet documentation for more information about writing classes.

- To learn how to write resource declarations, conditionals, and classes in a guided tour format, start at the beginning of Learning Puppet.
- For a complete but terse guide to the Puppet language's syntax, see the language guide.
- For complete documentation of the available resource types, see the type reference.
- For short printable references, see the modules cheat sheet and the core types cheat sheet.

Using a Homemade Module in the Console

- On the console, use the "add class" button to make the class available, just as in the previous example.
- Instead of assigning the class to a single node, assign it to a group. Navigate to the default group and use the edit button, then add the `core_permissions` class to its list of classes. Do not delete the existing classes, which are necessary for configuring new nodes.



- On the puppet master node, manually set dangerous permissions for the `crontab` and `passwd` files. This will make them editable by any unprivileged user.

```
# chmod 0666 /etc/crontab /etc/passwd
# ls -lah /etc/crontab /etc/passwd /etc/fstab
-rw-rw-rw- 1 root root  255 Jan  6  2007 /etc/crontab
-rw-r--r-- 1 root root  534 Aug 22  2011 /etc/fstab
-rw-rw-rw- 1 root root 2.3K Mar 26 08:18 /etc/passwd
```

- On the first agent node, manually set dangerous permissions for the `fstab` file:

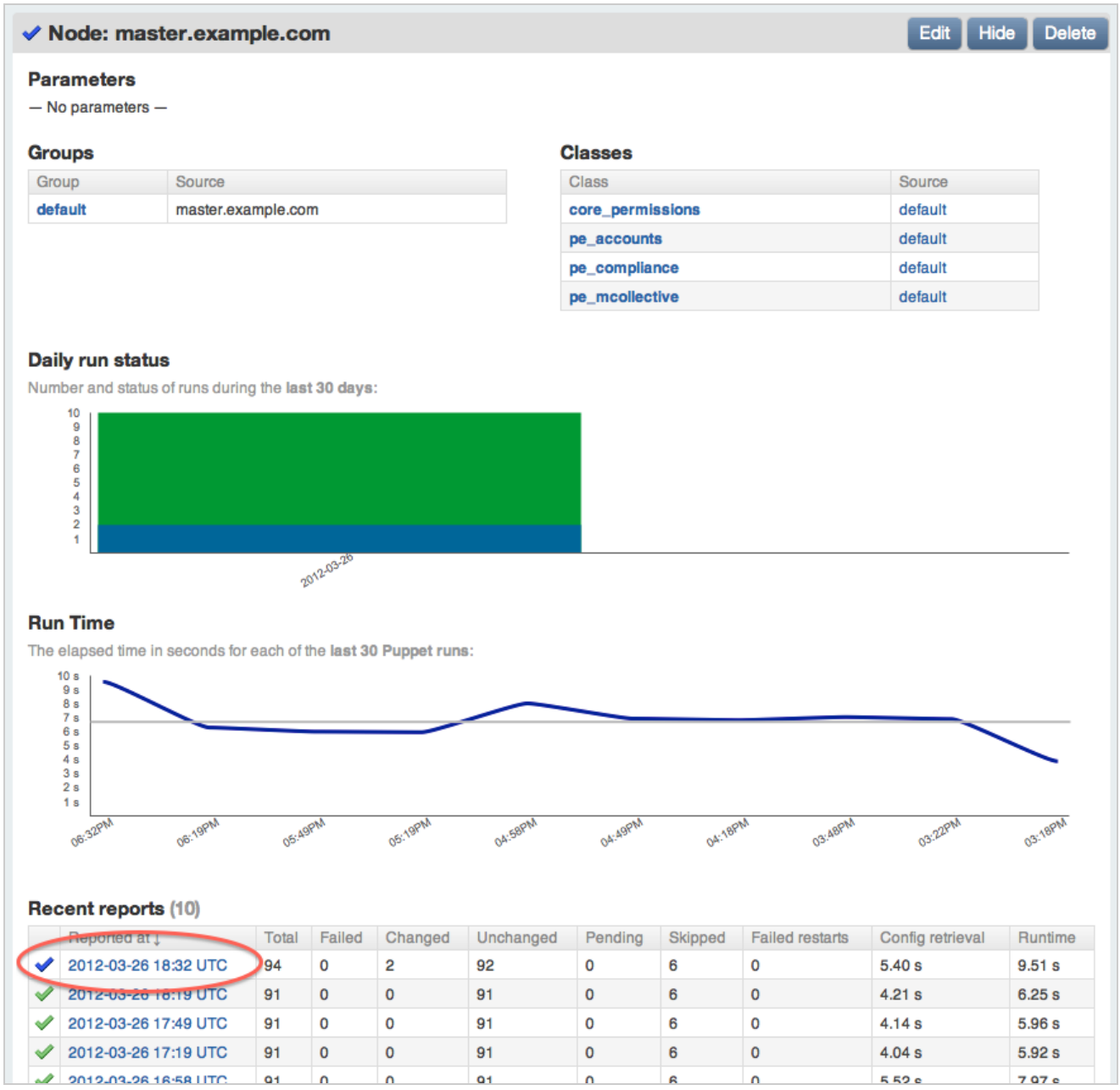
```
# chmod 0666 /etc/fstab
# ls -lah /etc/crontab /etc/passwd /etc/fstab
-rw-rw-r-- 1 root root  255 Jan  6  2007 /etc/crontab
-rw-rw-rw- 1 root root  534 Aug 22  2011 /etc/fstab
-rw-rw-r-- 1 root root 2.3K Mar 26 08:18 /etc/passwd
```

- Run puppet agent once on every node. You can do this by:
  - Doing nothing and waiting 30 minutes
  - Using live management to run Puppet on the two *nix nodes, then triggering a manual run on Windows
  - Triggering a manual run on every node, with either `puppet agent --test` or the “Run Puppet Agent” Start menu item

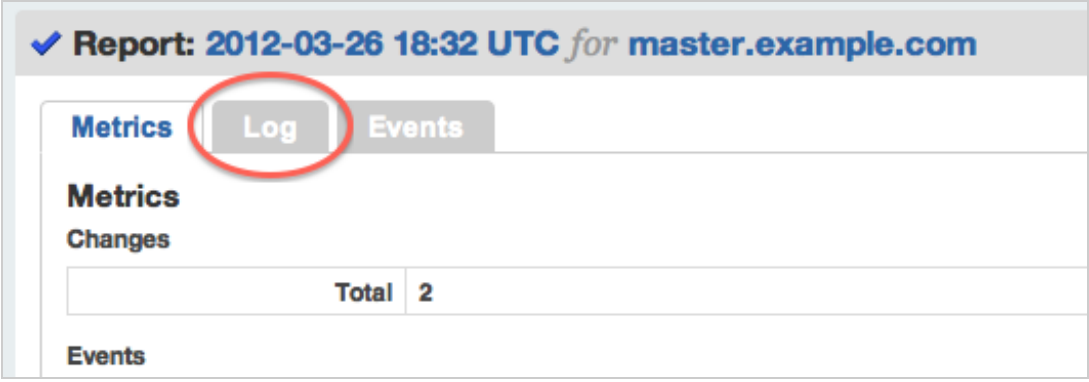
- On the master and first agent nodes, note that the permissions of the three files have been returned to safe defaults, such that only root can edit them:

```
# ls -lah /etc/fstab /etc/passwd /etc/crontab
-rw-r--r-- 1 root root  255 Jan  6  2007 /etc/crontab
-rw-r--r-- 1 root root  534 Aug 22  2011 /etc/fstab
-rw-r--r-- 1 root root 2.3K Mar 26 08:18 /etc/passwd
```

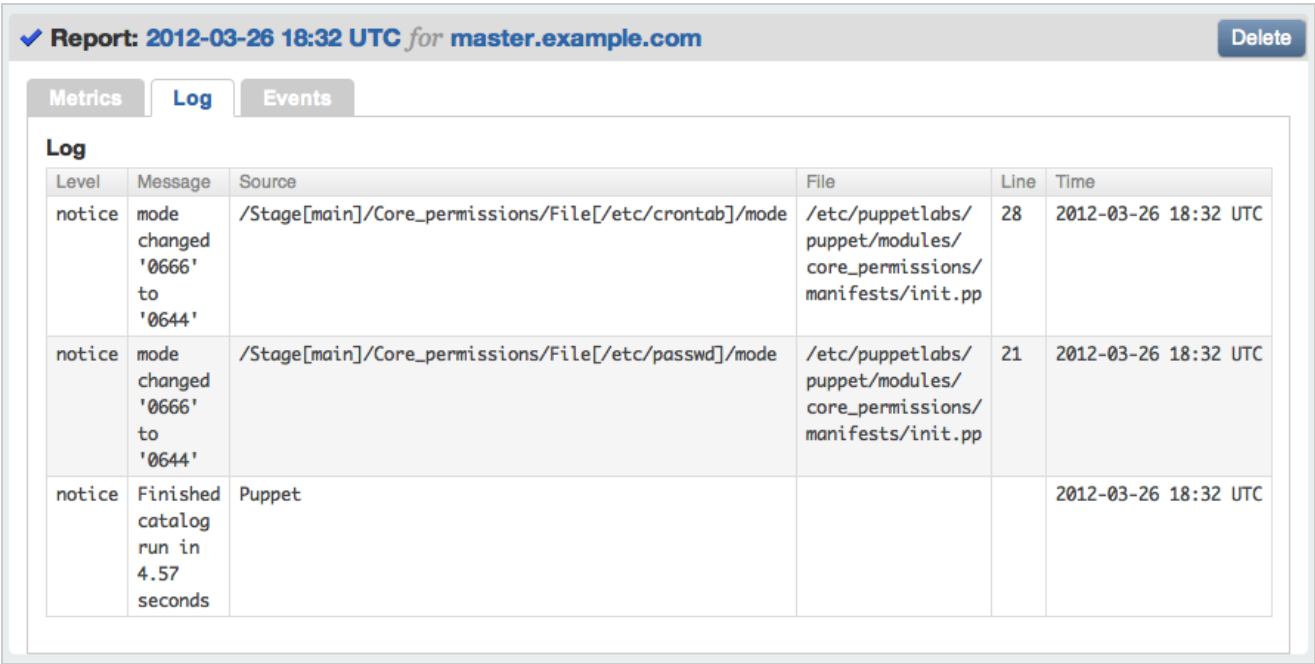
- On the Windows node, note that the class has safely done nothing, and has not accidentally created any files in `C:\etc\`.
- On the console, navigate to `master.example.com`, by clicking the nodes item in the top navigation and then clicking on the node's name. Scroll down to the list of recent reports, and note that the most recent one is blue, signifying that changes were made:



- Click on the topmost report, then navigate to the “log” tab of the report:



- Note the two changes made to the file permissions:



You have created a new class from scratch and used it to manage the security of critical files on your *nix servers.

Instead assigning it directly to nodes, you assigned it to a group. Using node groups can save you time and allow better visibility into your site. They are also crucial for taking advantage of the cloud provisioning tools. You can create new groups in the console with the “New group” button, and add new nodes to them using the “Edit” button on a group’s page.

Using a Site Module

Many users create a “site” module. Instead of describing smaller units of a configuration, the classes in a site module describe a complete configuration for a given type of machine. For example, a site module might contain:

- A `site::basic` class, for nodes that require security management but haven't been given a specialized role yet.
- A `site::webserver` class for nodes that serve web content.
- A `site::dbserver` class for nodes that provide a database server to other applications.

Site modules hide complexity so you can more easily divide labor at your site. System architects can create the site classes, and junior admins can create new machines and assign a single "role" class to them in the console. In this workflow, the console controls policy, not fine-grained implementation.

- On the puppet master, create the `/etc/puppetlabs/puppet/modules/site/manifests/basic.pp` file, and edit it to contain the following:

```
class site::basic {
  if $osfamily == 'windows' {
    include win_desktop_shortcut
  }
  else {
    include motd
    include core_permissions
  }
}
```

This class declares other classes with the `include` function. Note the "if" conditional that sets different classes for different OSes using the `$osfamily` fact. For more information about declaring classes, see the modules and classes chapters of Learning Puppet.

- On the console, remove all of the previous example classes from your nodes and groups, using the "edit" button in each node or group page. Be sure to leave the `pe_*` classes in place.
- Add the `site::basic` class to the console with the "add class" button in the sidebar.
- Assign the `site::basic` class to the default group.

> Your nodes are now receiving the same configurations as before, but with a simplified interface in the console. Instead of deciding which classes a new node should receive, you can decide what type of node it is and take advantage of decisions you made earlier.

Summary

You have now performed the core workflows of an intermediate Puppet user. In the course of their normal work, an intermediate user will:

- Download and hack Forge modules that almost (but not quite) fit their deployment's needs.
- Create new modules and write new classes to manage many types of resources, including files,

services, packages, user accounts, and more.

- Build and curate a site module to empower junior admins and simplify the decisions involved in deploying new machines.

- Next: System Requirements

System Requirements and Pre-Installation

Before installing Puppet Enterprise:

- Ensure that your nodes are running a supported operating system.
- Ensure that your puppet master and console servers are sufficiently powerful.
- Ensure that your network, firewalls, and name resolution are configured correctly.
- Plan to install the puppet master server before the console server, and the console server before any agent nodes.

Operating System

Puppet Enterprise 2.7 supports the following systems:

Operating system	Version	Arch	Roles
Red Hat Enterprise Linux	5 and 6	x86 and x86_64	all roles
CentOS	5 and 6	x86 and x86_64	all roles
Ubuntu LTS	10.04 and 12.04	32- and 64-bit	all roles
Debian	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
Microsoft Windows	2003, 2008, and 7	x86 and x86_864	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: 2-4 processor cores, at least 4 GB RAM, and very accurate timekeeping. Performance will vary, but this configuration can generally manage

approximately 1,000 agent nodes.

- The console role should usually be installed on a separate server from the puppet master, but can optionally be installed on the same server.
  - Minimum requirements: A machine able to handle moderate web traffic, perform processor-intensive background tasks, and run a disk-intensive MySQL database server. Requirements will vary significantly depending on the size and complexity of your site.

- The cloud provisioner role has very modest requirements.
  - Minimum requirements: A system which provides interactive shell access for trusted users. This system should be kept very secure, as the cloud provisioning tools must be given cloud service account credentials in order to function.

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

Configuration

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 agent nodes can use to 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.

Next

- To install Puppet Enterprise on *nix nodes, continue to Installing Puppet Enterprise.
- To install Puppet Enterprise on Windows nodes, continue to Installing Windows Agents.

Installing Puppet Enterprise

 This chapter covers *nix operating systems. To install PE on Windows, see Installing Windows Agents.

Downloading PE

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

Choosing an Installer Tarball

Puppet Enterprise can be downloaded in tarballs specific to your OS version and architecture, or as a universal tarball.

Note: The universal tarball is simpler to use, but is roughly ten times the size of a version-specific tarball.

AVAILABLE *NIX 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

Starting the Installer

- Unarchive the installer tarball, usually with `tar -xzf <TARBALL FILE>`.
- 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 select and configure PE's roles.
- 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`.

Using the Installer

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]`").

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.

Selecting Roles

First, the installer will ask which of PE's roles to install. The roles you choose will determine which other questions the installer will ask.

The Puppet Agent Role

This role should be installed on every node in your deployment, including the master and console nodes. (If you choose the puppet master or console roles, the puppet agent role will be installed automatically.) Nodes with the puppet agent role can:

- 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

In most deployments, this role should be installed on one node; installing multiple puppet masters requires additional configuration. The puppet master must be a robust, dedicated server; see the system requirements for more detail. The puppet master server can:

- Compile and serve configuration catalogs to puppet agent nodes.
- Route MCollective messages through its ActiveMQ server.
- Issue valid MCollective commands (from an administrator logged in as the `peadmin` user).

The Console Role

This role should be installed on one node. It should usually run on its own dedicated server, but can also run on the same server as the puppet master. The console server can:

- Serve 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.
- Collect reports from, and serve node information to the puppet master.

The Cloud Provisioner Role

This optional role can be installed on a computer where administrators have shell access. Since it requires confidential information about your cloud accounts to function, it should be installed on a secure system. Administrators can use the cloud provisioner tools 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.

Customizing Your Installation

After you choose the roles for the system you're installing onto, the installer will ask questions to configure those roles.

Puppet Master Questions

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 Questions

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 EMAIL AND PASSWORD

Access to the console's web interface is limited to approved users and governed by a lightweight system of roles. During installation, you must create an initial admin user for the console by providing an email address and password.

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

SMTP SERVER

The console's account management tools will send activation emails to new users, and requires an SMTP server to do so.

- If you cannot provide an SMTP server, an admin user can manually copy and email the activation

codes for new users. (Note that `localhost` will usually work as well.)

- If your SMTP server requires TLS or a user name and password, you must perform additional configuration after installing.

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.

DATABASES

The console needs multiple MySQL databases and MySQL users in order to operate, but can automatically configure them. It can also automatically install the MySQL server if it isn't already present on the system.

The installer gives slightly different options to choose from depending on your system's configuration:

- Automatically install a MySQL server and auto-configure databases (only available if MySQL is not yet installed). This option will generate a random root MySQL password, and you will need to look it up in the saved answer file after installation finishes. A message at the end of the installer will tell you the location of the answer file.
- Auto-configure databases on an existing local or remote MySQL server. You will need to provide your server's root MySQL password to the installer. (Note that if you want to auto-configure databases on a remote server, you must make sure the root MySQL user is allowed to log in remotely.)
- Use a set of pre-existing manually configured databases and users.

MANUAL DATABASE CONFIGURATION

If you are manually configuring your databases, the installer will ask you to provide:

- A name for the console's primary database
- A MySQL user name for the console
- A password for the console's user
- A name for the console authentication database
- A MySQL user name for console authentication
- A password for the console authentication user
- The database server's hostname (if remote)
- The database server's port (if remote; default: 3306)

You will also need to make sure the databases and users actually exist. 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';

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

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

Consult the MySQL documentation for more info.

Puppet Agent Questions

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`.

OPTIONAL: OVERRIDE FAILED PUPPET MASTER COMMUNICATIONS

The installer will attempt to contact the puppet master before completing the installation. If it isn't able to reach it, it will give you a choice between correcting the puppet master hostname, or continuing anyway.

Final Questions

Vendor Packages

Puppet Enterprise may need some extra system software from your OS vendor's package repositories.

- The puppet master role requires a Java runtime, in order to run the ActiveMQ server for orchestration.
- The console role requires MySQL; if using local databases, it also requires MySQL server.

If these aren't already present, the installer will offer to automatically install them. If you decline, it will exit, and you will need to install them manually before running the installer again.

JAVA AND MYSQL VERSIONS

- On every supported platform, PE can use the default system packages for MySQL and Java (OpenJDK on most Linuxes, and IBM Java on SUSE).
- Custom-compiled MySQL or Java versions may or may not work, as Puppet Enterprise expects to find shared objects and binaries in their standard locations. In particular, we have noticed problems with custom compiled MySQL 5.5 on Enterprise Linux variants.
- On Enterprise Linux variants, you may optionally use the Java and MySQL packages provided by Oracle. Before installing PE, you must manually install Java and/or MySQL, then 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.

Note: If installing `pe-virtual-java`, make sure that the `keytool` binary is in one of the following directories:

- `/opt/puppet/bin`
- `/usr/kerberos/sbin`
- `/usr/kerberos/bin`
- `/usr/local/sbin`
- `/usr/local/bin`
- `/sbin`
- `/bin`
- `/usr/sbin`
- `/usr/bin`

If `keytool` isn't already there, use `find` or `which` to locate it, and symlink it into place so that the PE installer can find it during installation. This binary is necessary for configuring MCollective.

```
$ which keytool
/path/to/keytool
$ sudo ln -s /path/to/keytool /usr/local/bin/keytool
```

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, pe-man`, and `mco` binaries.

Confirming Installation

The installer will offer a final chance to confirm your answers before installing.

After Installing

Securing the Answer File

After finishing, the installer will print a message telling you where it saved its answer file. If you automatically configured console databases, you should save and secure this file, as it will contain the randomly-generated root MySQL password.

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.

Node requests can be approved or rejected using the console's certificate management capability. Pending node requests are indicated in the main nav bar. Click on the "node requests" indicator to go to a view where you can see current requests and approve or reject them as needed.

Alternatively, you can use the CLI. Certificate signing with the CLI 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. You can trigger a puppet run manually on the node if you want to see it right away.

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.

Note that you can download and install Puppet Enterprise on up to ten nodes at no charge. No licence key is needed to run PE on up to ten nodes.

- Next: Upgrading

Installing Windows Agents

 This chapter refers to Windows functionality. To install PE on *nix nodes, see Installing Puppet Enterprise.

PE 2.7 includes support for Windows agent nodes. Windows nodes:

- Can fetch configurations from a puppet master and apply manifests locally
- Cannot serve as a puppet master
- Cannot respond to live management or orchestration

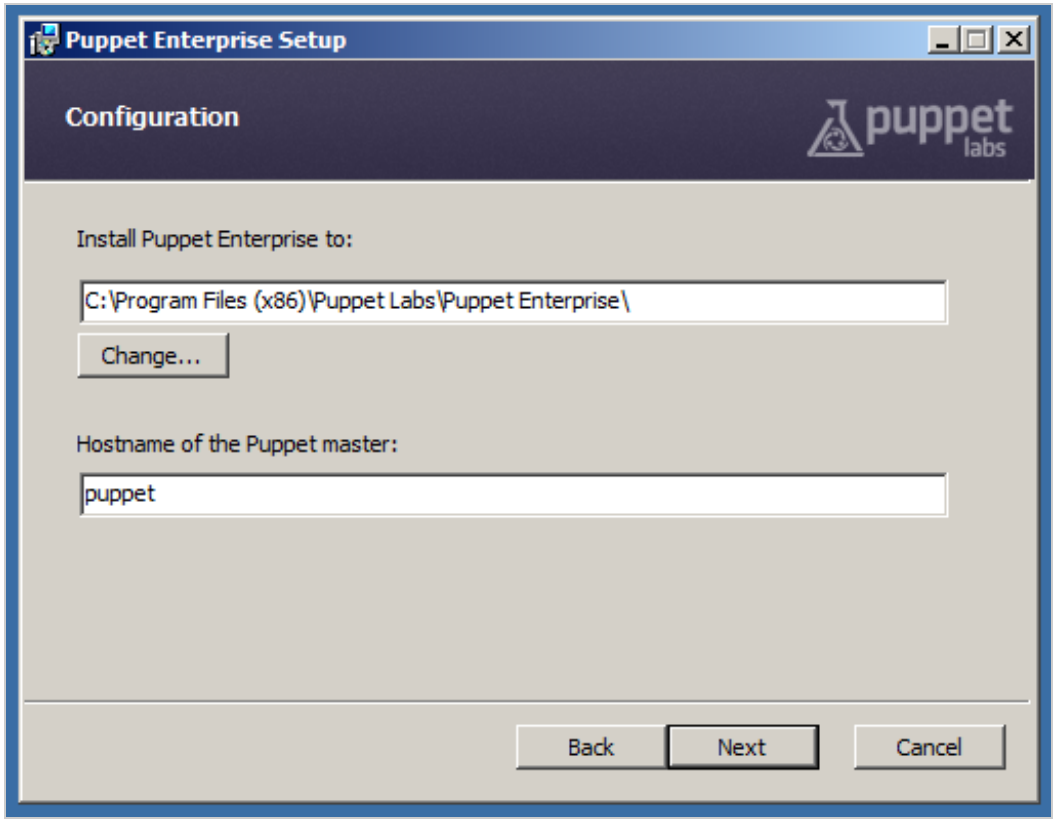
See the main Puppet on Windows documentation for details on running Puppet on Windows and writing manifests for Windows.

Installing Puppet

To install Puppet Enterprise on a Windows node, simply download and run the installer, which is a standard Windows .msi package and will run as a graphical wizard.

The installer must be run with elevated privileges. Installing Puppet does not require a system reboot.

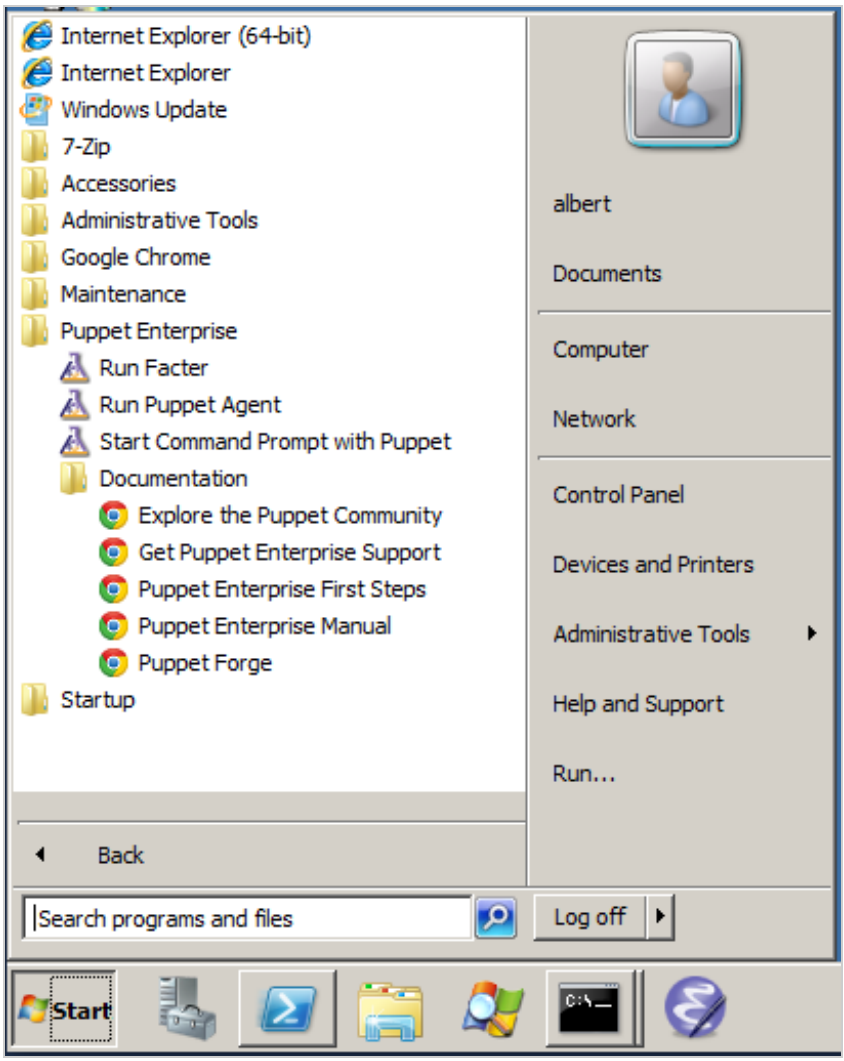
The only information you need to specify during installation is the hostname of your puppet master server:



After Installation

Once the installer finishes:

- Puppet agent will be running as a Windows service, and will fetch and apply configurations every 30 minutes; you can now assign classes to the node on your puppet master or console server. Puppet agent can be started and stopped with the Service Control Manager or the `sc.exe` utility; see Running Puppet on Windows for more details.
- The Start Menu will contain a Puppet folder, with shortcuts for running puppet agent manually, running Facter, and opening a command prompt for use with the Puppet tools. See Running Puppet on Windows for more details.



Automated Installation

For automated deployments, Puppet can be installed unattended on the command line as follows:

```
msiexec /qn /i puppet.msi
```

You can also specify `/l*v install.txt` to log the progress of the installation to a file.□

The following public MSI properties can also be specified:□

MSI Property	Puppet Setting	Default Value
`INSTALLDIR`	n/a	Version-dependent, see below
`PUPPET_MASTER_SERVER`	`server`	`puppet`
`PUPPET_CA_SERVER`	`ca_server`	Value of `PUPPET_MASTER_SERVER`
`PUPPET_AGENT_CERTNAME`	`certname`	Value of `facter fdqn` (must be lowercase)

For example:

```
msiexec /qn /i puppet.msi PUPPET_MASTER_SERVER=puppet.acme.com
```

Upgrading

Puppet can be upgraded by installing a new version of the MSI package. No extra steps are required, and the installer will handle stopping and re-starting the puppet agent service.

When upgrading, the installer will not replace any settings in the main puppet.conf configuration file, but it can add previously unspecified settings if they are provided on the command line.

Uninstalling

Puppet can be uninstalled through Windows' standard "Add or Remove Programs" interface, or from the command line.

To uninstall from the command line, you must have the original MSI file or know the ProductCode of the installed MSI:

```
msiexec /qn /x [puppet.msi|product-code]
```

Uninstalling will remove Puppet's program directory, the puppet agent service, and all related registry keys. It will leave the data directory intact, including any SSL keys. To completely remove Puppet from the system, the data directory can be manually deleted.

Installation Details

What Gets Installed

In order to provide a self-contained installation, the Puppet installer includes all of Puppet's dependencies, including Ruby, Gems, and Facter. (Puppet redistributes the 32-bit Ruby application from rubyinstaller.org.

These prerequisites are used only for Puppet and do not interfere with other local copies of Ruby.

Program Directory

Unless overridden during installation, Puppet and its dependencies are installed into the standard Program Files directory for 32-bit applications.

Puppet Enterprise's the default installation path is:

OS type	Default Install Path
32-bit	`C:\Program Files\Puppet Labs\Puppet Enterprise`
64-bit	`C:\Program Files (x86)\Puppet Labs\Puppet Enterprise`

The program files directory can be located using the `PROGRAMFILES` environment variable on 32-bit versions of Windows, or the `PROGRAMFILES(X86)` variable on 64-bit versions.

Puppet's program directory contains the following subdirectories:

Directory	Description
bin	scripts for running Puppet and Facter
facter	Facter source
misc	resources
puppet	Puppet source
service	code to run puppet agent as a service
sys	Ruby and other tools

Data Directory

Puppet stores its settings (`puppet.conf`), manifests, and generated data (like logs and catalogs) in its data directory.

When run with elevated privileges — Puppet's intended state — the data directory is located in the `COMMON_APPDATA` folder. This folder's location varies by Windows version:

OS Version	Path	Default
2003	`%ALLUSERSPROFILE%\Application Data\PuppetLabs\puppet`	`C:\Documents and Settings\All Users\Application Data\PuppetLabs\puppet`
7, 2008	`%PROGRAMDATA%\PuppetLabs\puppet`	`C:\ProgramData\PuppetLabs\puppet`

Since CommonAppData directory is a system folder, it is hidden by default. See http://support.microsoft.com/kb/812003 for steps to show system and hidden files and folders.

If Puppet is run without elevated privileges, it will use a `.puppet` directory in the current user's home folder as its data directory. This may result in Puppet having unexpected settings.

Puppet's data directory contains two subdirectories:

- `etc` contains configuration files, manifests, certificates, and other important files
- `var` contains generated data and logs

More

For more details about using Puppet on Windows, see:

- Running Puppet on Windows
- Writing Manifests for Windows

- Next: Upgrading

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.

 To upgrade Windows nodes, simply download and run the new MSI package as described in Installing Windows Agents.

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.

Warning: If you have created custom modules and stored them in `/opt/puppet/share/puppet/modules`, the upgrader will fail. Before upgrading, you should move your custom modules to `/etc/puppetlabs/puppet/modules`. Alternatively, you can update your modules manually to have the correct metadata.

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 Installing PE of this guide for information on downloading Puppet Enterprise tarballs.

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 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.

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 2.7 includes a cloud provisioner tool that can be installed on trusted nodes where administrators have shell access. On nodes which lack the cloud provisioner role, you'll be asked whether you wish to install it.

Vendor Packages

If PE 2.7 needs any packages from your OS's repositories, it will ask permission to install them.

Puppet Master Options

REMOVING `MCO`'S HOME DIRECTORY

When upgrading from PE 1.2, the `mco` user 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

When upgrading from PE 1.2, the `mcollectivepe`, `accounts`, and `baselines` modules will be 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

ADMIN USER EMAIL AND PASSWORD

The console now uses role-based user authentication. You will be asked for an email address and password for the initial admin user; additional users can be configured after the upgrade is completed.

Upgrader Warnings

On console servers, the upgrader will check your MySQL server's `innodb_buffer_pool_size` setting. If it is too small, the upgrader will advise you to increase it.

If you receive a warning about the `innodb_buffer_pool_size` setting, you should:

- Cancel the upgrade and exit the upgrader.
- Follow these instructions to increase the buffer size.
- Re-run the upgrader and allow it to finish.

Final Steps: From PE 2.0 or 1.2

- If you received an upgrader warning on your console server as described above, be sure to increase your MySQL server's `innodb_buffer_pool_size`.

Otherwise, no extra steps are needed when upgrading from PE 2.0 or 1.2.

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 2.7, 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
```

- Next: 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.

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 uninstallation 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
```

- Next: Automated Installation

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.

- Next: Answer File Reference →

Answer File Reference

Answer files are used for automated installations 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`

`q_install_cloud_provisioner`
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.

`q_fail_on_unsuccessful_master_lookup`
Y or N — Whether to quit the install if the puppet master cannot be reached.

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_email`
String — The email address with which the console's admin user will log in. Note that this answer has changed from PE 2.0.

`q_puppet_enterpriseconsole_auth_password`
String — The password for the console's admin user. Must be longer than eight characters.

`q_puppet_enterpriseconsole_smtp_host`
String – The SMTP server with which to email account activation codes to new console users.

`q_puppet_enterpriseconsole_inventory_certname`
String — An identifying 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.

`q_puppet_enterpriseconsole_setup_auth_db`
Y or N — Whether to automatically configure the console authentication database.

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

`q_puppet_enterpriseconsole_auth_database_user`
String — The MySQL user the console authentication 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 auth database.

`q_puppet_enterpriseconsole_auth_database_password`
String — The password for the auth database'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_setup_auth_db`
Y or N — Whether to automatically configure the console's authentication database.

`q_puppet_enterpriseconsole_database_root_password`
String — The password for the root user on the console's database server. Only required if `q_puppet_enterpriseconsole_setup_auth_db` is true.

`q_puppet_enterpriseconsole_auth_user_email`
String — The email address with which the console's admin user will log in. Note that this answer has changed from PE 2.0. Only required if this node has the console role installed.

`q_puppet_enterpriseconsole_auth_password`
String — A password for accessing the console. Previous versions of PE 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_database_name`
String — The database the console authentication will use. Note that if you are not automatically configuring the auth database, this database must already exist on the MySQL server.

`q_puppet_enterpriseconsole_auth_database_user`
String — The MySQL user the console authentication 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 auth database.

`q_puppet_enterpriseconsole_auth_database_password`
String — The password for the auth database's MySQL user.
`q_puppet_enterpriseconsole_smtp_host`
String – The SMTP server with which to email account activation codes to new console users.
`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.

- Next: What gets installed where?

PE 2.7 » Installing » 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.

Software

What

All working components of PE, excluding configuration files. These are files you are not likely to need to change.

Where

All PE software (excluding config files) 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.

Configuration Files

What

Files used to configure Puppet and its subsidiary components. These are the files you will likely change to accomodate the needs of your environment.

Where

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`.

Log Files

What

The software distributed with Puppet Enterprise generates the following log files, which can be found as follows.

Where

Puppet Master Logs

- `/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 Logs

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 Logs

- `/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 Log

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

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

Console Log

- `/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 Logs

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`

Puppet Enterprise Components

Tools

Puppet Enterprise installs several suites of command line tools to help you work with the major components of the software. These include:

- Puppet Tools: Tools that control basic functions of Puppet such as `puppet master`, `puppet apply` and `puppet cert`. See the Tools Section of the Puppet Manual for more information.
- Cloud Provisioning Tools: Tools used to provision new nodes. Mostly based around the `node` subcommand, these tools are used for tasks such as creating or destroying virtual machines, classifying new nodes, etc. See the Cloud Provisioning Section for more information.
- Orchestration Tools: Tools used to orchestrate simultaneous actions across a number of nodes. These tools are built on the MCollective framework and are accessed either via the `mco` command or via the Live Management tab of the PE console. See the Orchestration Section for more information.
- Module Tools: The Module tool is used to access and create Puppet Modules, which are reusable chunks of Puppet code users have written to automate configuration and deployment tasks. For more information, and to access modules, visit the Puppet Forge.
- Console: The console is Puppet Enterprise's GUI web interface. The console provides tools to view and edit resources on your nodes, view reports and activity graphs, trigger Puppet runs, etc. See the Console Section of the Puppet Manual for more information.

For more details, you can also refer to the man page for a given command or subcommand.

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 section 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.

Certificates

PE generates a number of certificates at install. These are:

- `pe-internal-dashboard` — The certificate for the puppet dashboard.
- `q_puppet_enterpriseconsole_install` — The certificate for the PE console. Only generated if the user has chosen to install the console.
- `q_puppetmaster_install` — This certificate is either generated at install if the puppet master and console are the same machine or is signed by the master if the console is on a separate machine.
- `pe-internal-mcollective-servers` — A shared certificate generated on the puppet master and shared to all Mcollective servers.
- `pe-internal-peadmin-mcollective-client` — The certificate for the peadmin account on the puppet master.
- `pe-internal-puppet-console-mcollective-client` — The certificate for the PE Console/Live Management
- `pe-internal-broker` — The certificate generated for the activemq instance running over SSL on the puppet master. Added to /etc/puppetlabs/activemq/broker.ks.

A fresh PE install should thus give the following list of certificates:

```
root@master:~# puppet cert list --all
+ "master"
(40:D5:40:FA:E2:94:36:4D:C4:8C:CE:68:FB:77:73:AB) (alt names: "DNS:master",
"DNS:puppet", "DNS:puppet.soupkitchen.internal")
+ "pe-internal-broker"
(D3:E1:A8:B1:3A:88:6B:73:76:D1:E3:DA:49:EF:D0:4D) (alt names: "DNS:master",
"DNS:master.soupkitchen.internal", "DNS:pe-internal-broker", "DNS:stomp")
+ "pe-internal-dashboard"
(F9:10:E7:7F:97:C8:1B:2F:CC:D9:F1:EA:B2:FE:1E:79)
+ "pe-internal-mcollective-servers"
(96:4F:AA:75:B5:7E:12:46:C2:CE:1B:7B:49:FF:05:49)
+ "pe-internal-peadmin-mcollective-client"
(3C:4D:8E:15:07:41:18:E2:21:57:19:01:2E:DB:AB:07)
+ "pe-internal-puppet-console-mcollective-client"
(97:10:76:B5:3E:8D:02:D2:3D:A6:43:F4:89:F4:8B:94)
```

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 no longer functional. Use the above method instead.

- Next: Accessing the Console

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
- Manage console users and their access privileges

 NOTE: Live management and MCollective are not yet supported on Windows nodes.

Browser Requirements

The following browsers are supported for use with the console:

- Chrome (current versions)
- Firefox 3.5 and higher
- Safari 4 and higher
- Internet Explorer 8 and higher

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 can reach the console by navigating to:

`https://console.domain.com`

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

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

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

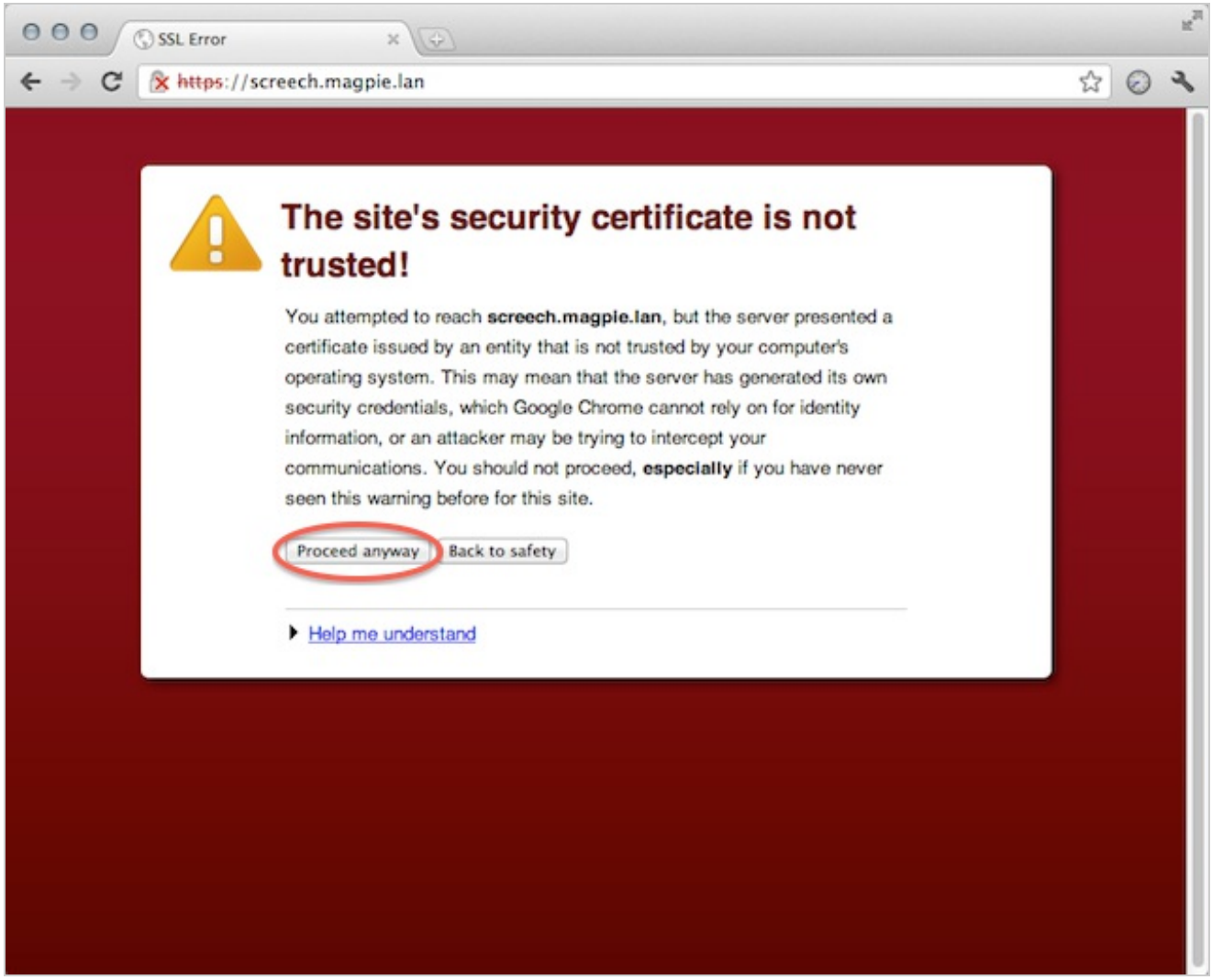
Accepting the Console's Certificate

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

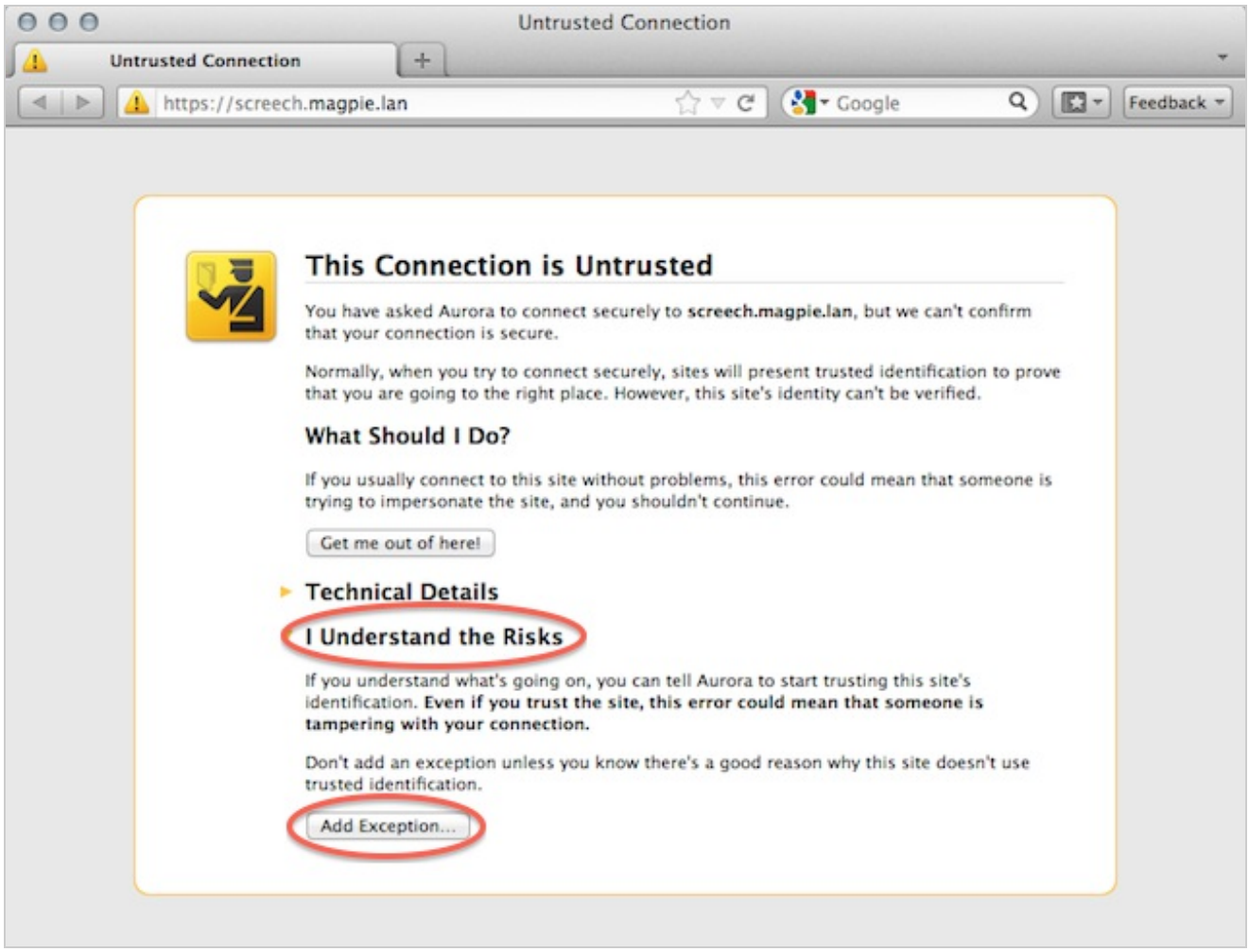
Use the "Proceed anyway" button on Chrome's warning page.



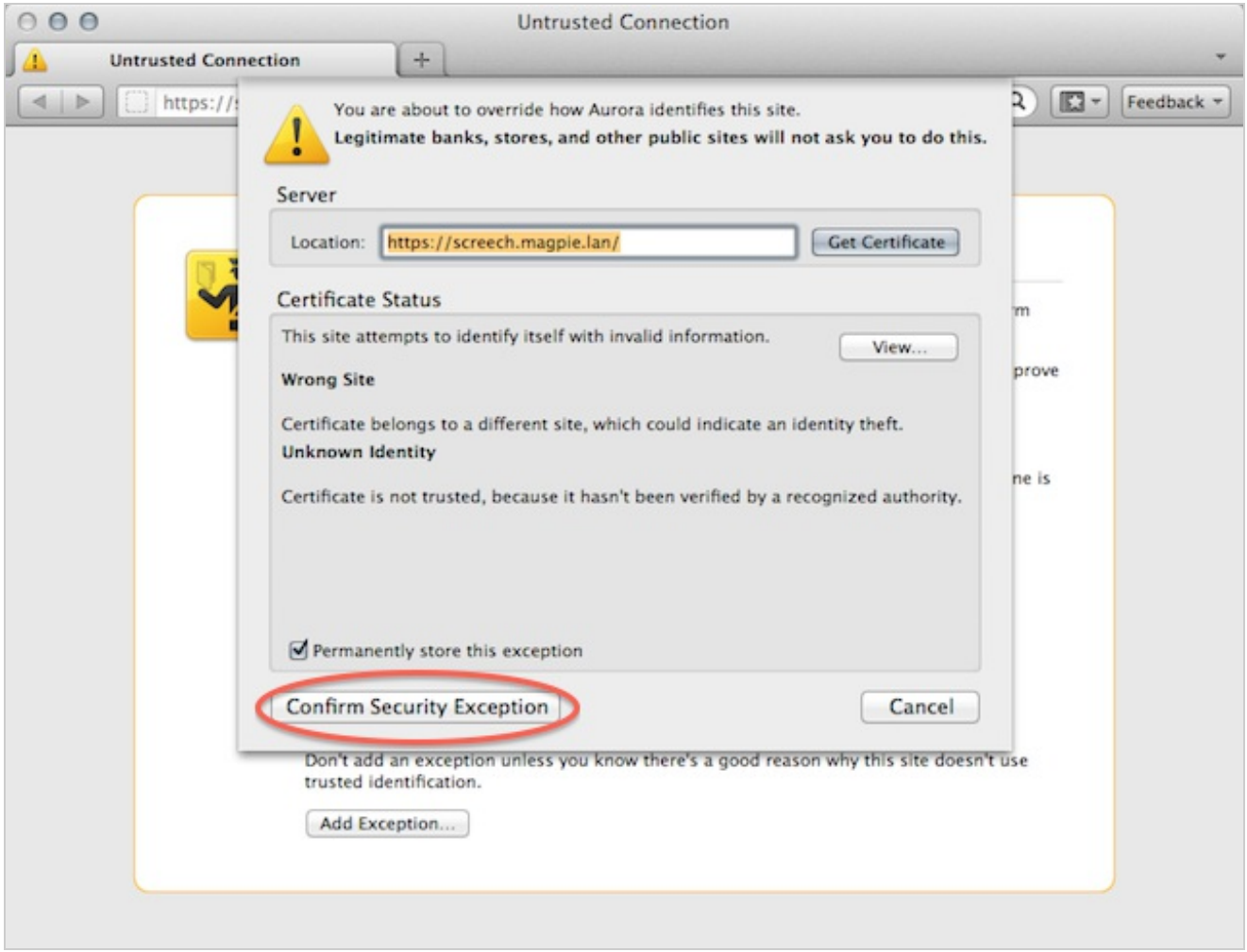
Accepting the Certificate in Mozilla Firefox

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:

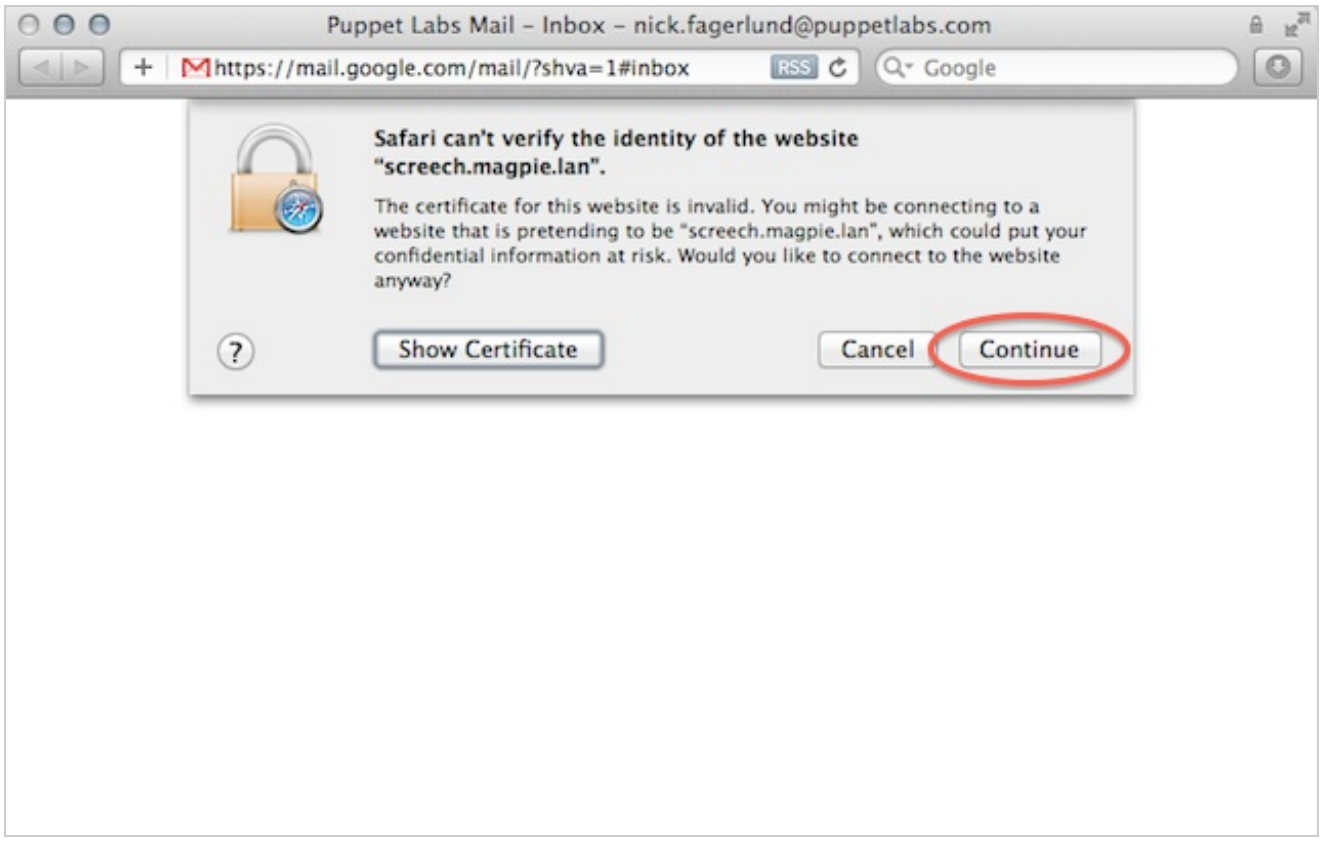

Untrusted Connection
Untrusted Connection
https://screech.magpie.lan
Google
Feedback
This Connection is Untrusted
You have asked Aurora to connect securely to screech.magpie.lan, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted 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:



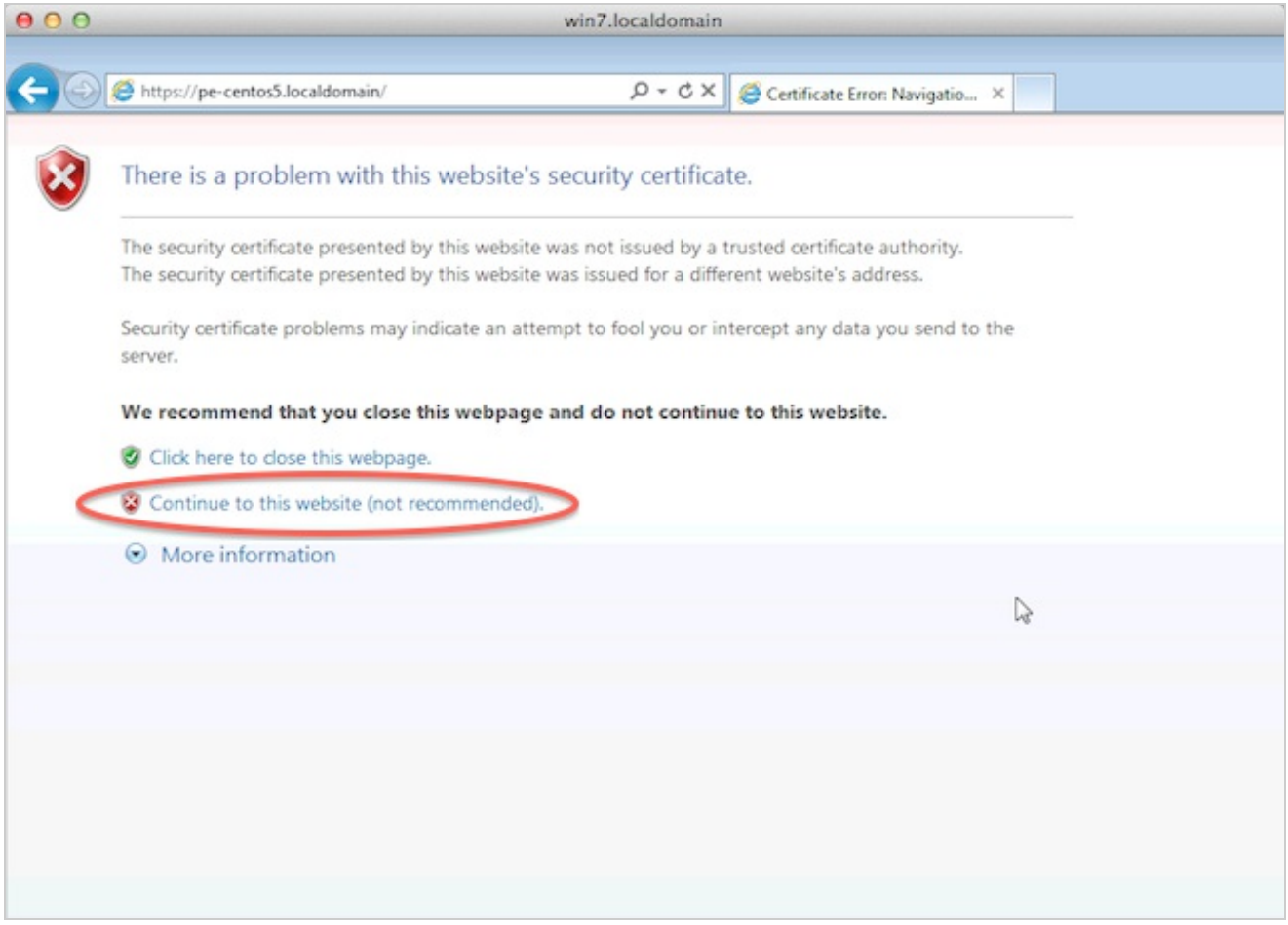
Accepting the Certificate in Apple Safari

Click the “Continue” button on the warning dialog.



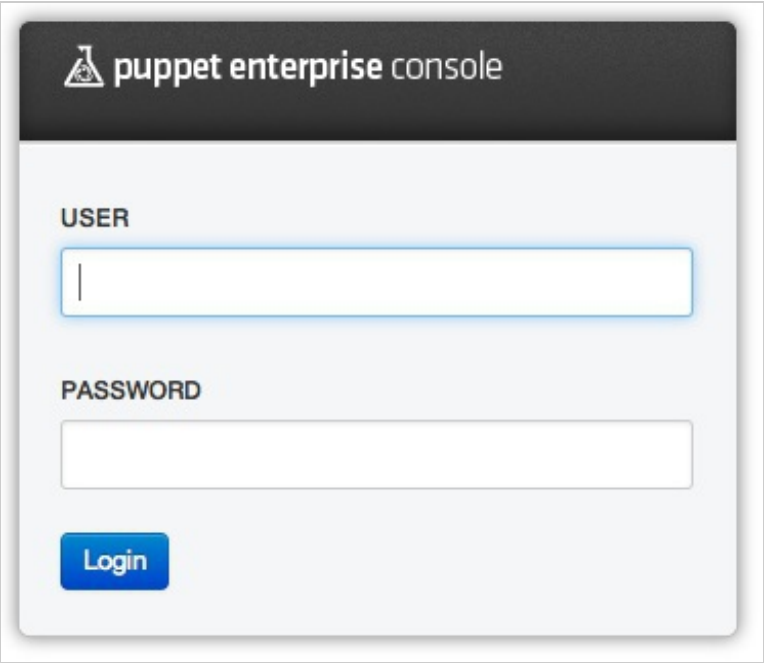
Accepting the Certificate in Microsoft Internet Explorer

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



Logging In

For security, accessing the console requires a user name and password. In PE 2.7, there are three different levels of user access: read, read-write, and admin. If you are an admin setting up the console or accessing it for the first time, use the user and password you chose when you installed the console. Otherwise, you will need to get credentials from your site's administrator. See the User Management page for more information on managing console user accounts.



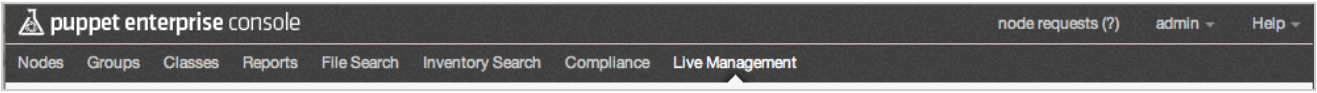
Since the console is the main point of control for your infrastructure, you will probably want to decline your browser's offer to remember its password.

- Next: Navigating the Console

Navigating the Console

Getting Around

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



All of the menu items deal with aspects of your nodes and their configuration except:

- Help which provides help, obviously.
- Your username which provides access to your account information and, if you are an admin user, also provides access to user management tools. For information on the user management tools, see the User Management and Authentication page.

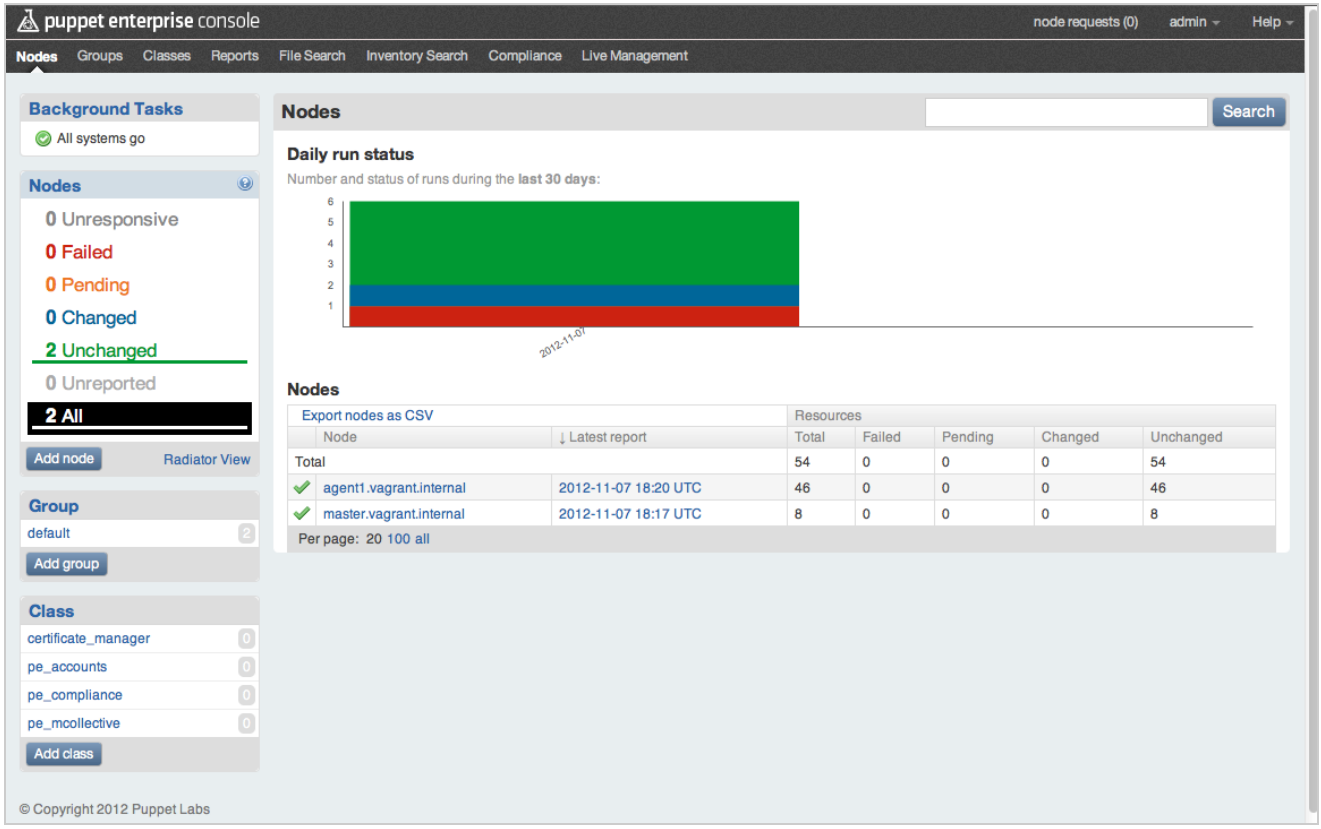
Note: For users limited to read-only access, some elements of the console shown here will not be visible.

What's in the Console?

The console deals with three main objects:

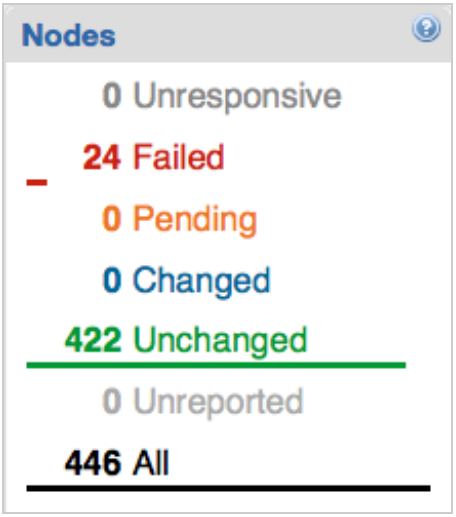
- A node represents a single system being managed by Puppet. It can have classes applied to it, and can be a member of groups.
- A group is an arbitrary set of nodes. It can have classes applied to it, and can contain nodes.
- A class represents a Puppet class available in your puppet master's collection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

Nodes



You can see nodes requesting to join your site by clicking on the "node requests" indicator at the top right of the main navigation bar. After a node completes its first Puppet run (which may take up to 30 minutes after you've approved its request), it will appear in the console and can be added to groups and have classes applied to it.

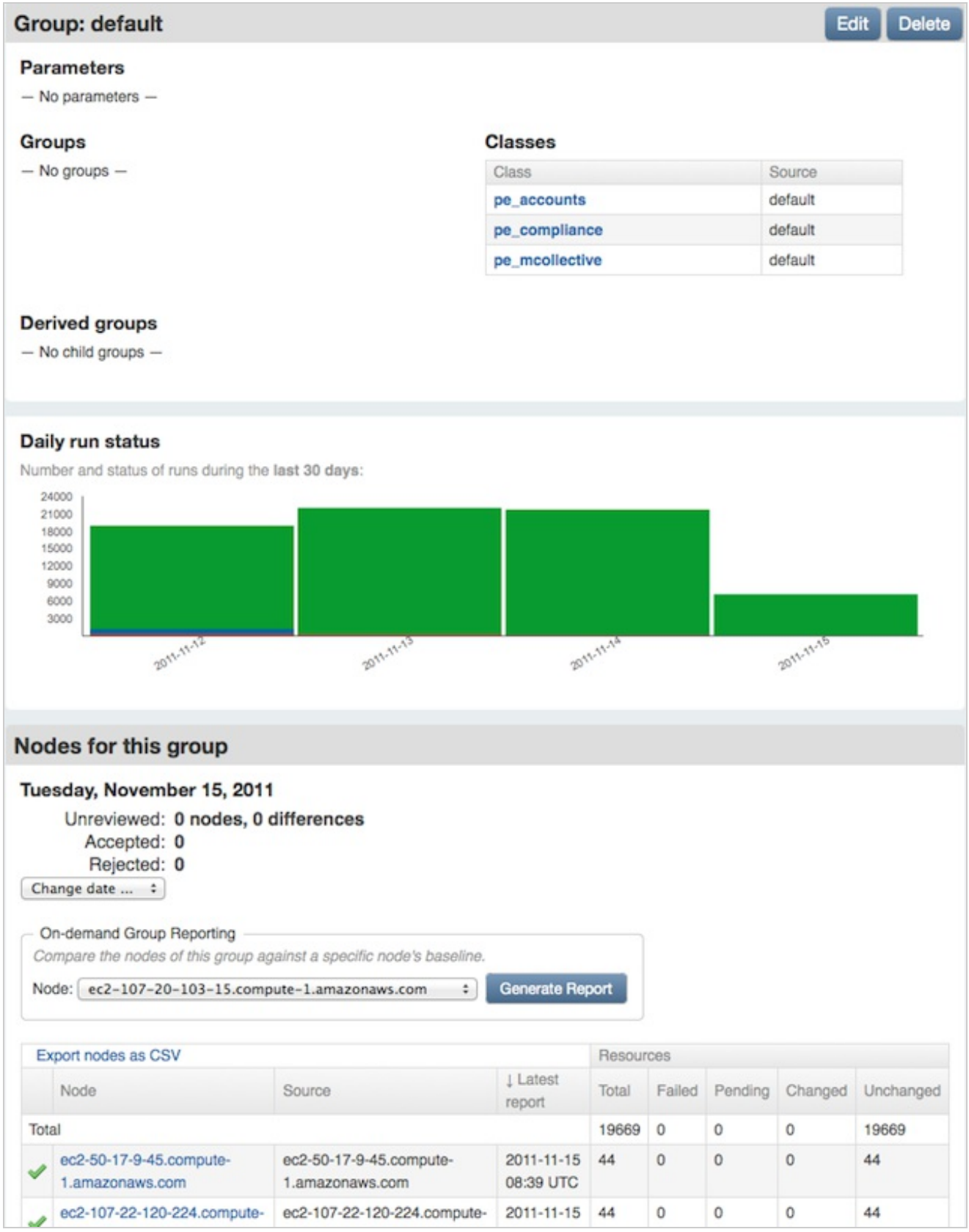
Since the console receives a report every time Puppet runs on a node, it keeps a running count in the sidebar of what state your nodes are in:



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. Click 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 all set; Puppet will apply it as needed, and you can click the class in the console to see which nodes it's been assigned to.

- Next: Responding to Node Requests with Certificate Management

Working with Node Requests

Intro/Overview

In previous versions of PE, sysadmins had to use the command line on the puppet master to manually approve or reject certificates from agent nodes attempting to join the site. Node request management is a new capability of the PE console which allows sysadmins to view and respond to requests graphically, from within the console. This means nodes can now be approved without needing access to the puppet master. For further security, node request management supports the console's user management system: only users with read/write privileges can take action on node requests.

Once the console has been properly configured to point at the appropriate Certificate Authority, it will display all of the nodes that have generated Certificate Signing Requests (CSRs). You can then approve or deny the requests, individually or in a batch.

For each node making a request, you can also see its name and associated CSR fingerprint.

Viewing Node Requests

You can view the number of pending node requests from anywhere in the console by checking the indicator in the top right of the main menu bar.

Click on the pending nodes indicator to view and manage the current requests.

You will see a view containing a list of all the pending node requests. Each item on the list shows the node's name and its corresponding CSR's fingerprint. (Click on the truncated fingerprint to view the whole thing in a pop-up.)

If there are no pending node requests, you will see some instructions for adding new nodes. If this is not what you expect to see, the location of your Certificate Authority (CA) may not be configured correctly.

Rejecting and Approving Nodes

The ability to respond to node requests is linked to your user privileges. You must be logged in to the console as a user with read/write privileges before you can respond to requests.

Use the buttons to accept or reject nodes, singly or all at once. Note that once a node request is approved, the node will not show up in the console until the next puppet run takes place. This could be as long as 30 minutes, depending on how you have set up your puppet master. Depending

on how many nodes you have in your site total, and on the number of pending requests, it can also take up to two seconds per request for “Reject All” or “Accept All” to finish processing.

In some cases, DNS altnames may be set up for agent nodes. In such cases, you cannot use the console to approve/reject node requests. The CSR for those nodes must be accepted or rejected using `puppet cert` on the CA. For more information, see the DNS altnames entry in the reference guide.

In some cases, attempting to accept or reject a node request will result in an error. This is typically because the request has been modified somehow, usually by being accepted or rejected elsewhere (e.g. by another user or from the CLI) since the request was first generated.

Accepted/rejected nodes will remain displayed in the console for 24 hours after the action is taken. This interval cannot be modified.

WORKING WITH REQUESTS FROM THE CLI

You can still view, approve, and reject node requests using the command line interface.

You can view pending node requests in the CLI by running

```
$ sudo puppet cert list
```

To sign one of the pending requests, run:

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

For more information on working with certificates from the CLI, see the Puppet tools guide or view the man page for `puppet cert`.

Configuration Details

- By default, the location of the CA is set to the location of PE’s puppet master. If the CA is in a custom location (as in cases where there are multiple puppet masters), you will have to set the `ca_server` and `ca_port` parameters in the `/opt/puppet/share/puppet-dashboard/config/settings.yml` file.
- When upgrading PE to 2.7, the upgrader will convert the currently installed auth.conf file to one that is fully managed by Puppet and which includes a new rule for request management. If auth.conf has been manually modified prior to the upgrade, the upgrader will not convert the file. Consequently, you will need to add the new rule manually by adding the code below into `/etc/puppetlabs/puppet/auth.conf`:

```
    path  /certificate_status
```

```
    method find, search
    auth yes
    allow pe-internal-dashboard
```

Also, note that since auth.conf is fully managed by puppet in 2.7, if you make changes to the file, they will get over-written on the next puppet run.

NEW MODULES

PE 2.7 installs three modules needed for node request management: `puppetlabs-request_manager`, `puppetlabs-auth_conf`, and `ripienar-concat`. It also upgrades the `puppetlabs-stdlib` module to v.2.5.1.

The `puppetlabs-auth_conf` module contains a new defined type: `auth_conf::acl`. The type takes the following parameters:

parameter	description	value types	default value	required
path	URL path of ACL	string	$title	no
acl_method	find, search save, delete	string, array		no
auth	yes, no, any	sring	yes	no
allow	certnames to access path	array	[ ]	no
order	order in auth.conf file	string	99	no
regex	is the path a regex?	bool	false	no
environment	environments to allow	string		no

- Next: Viewing Reports and Inventory Data

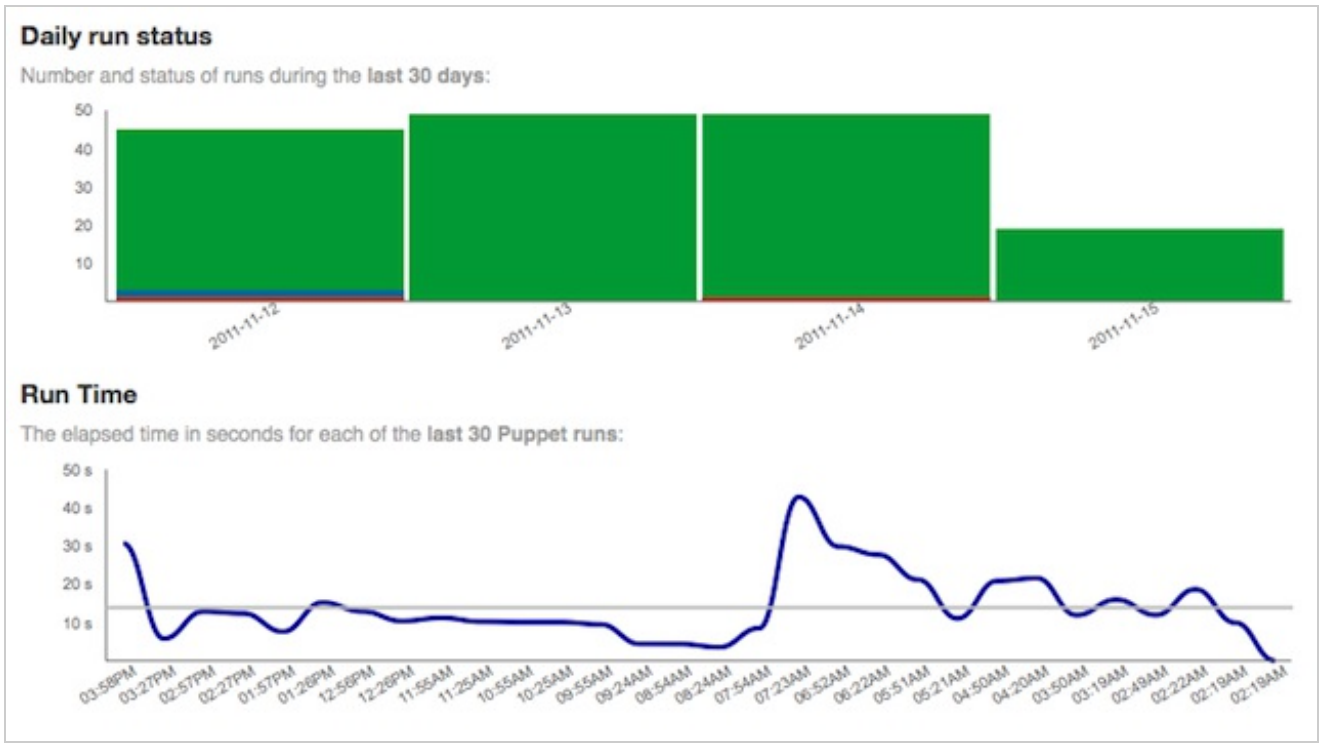
Viewing Reports and Inventory Data

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.

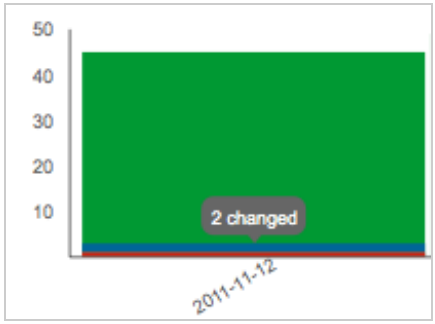
Reading Reports

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.

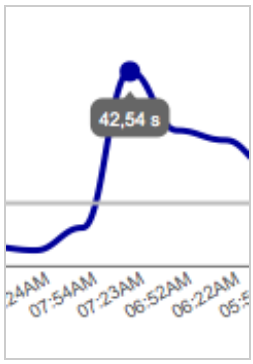


The daily run status histogram is broken down with the same colors that indicate run status in the console's sidebar: red for failed runs, orange for pending runs (where a change would have been made, but the resource to be changed was marked as no-op), blue for successful runs where changes were made, 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:



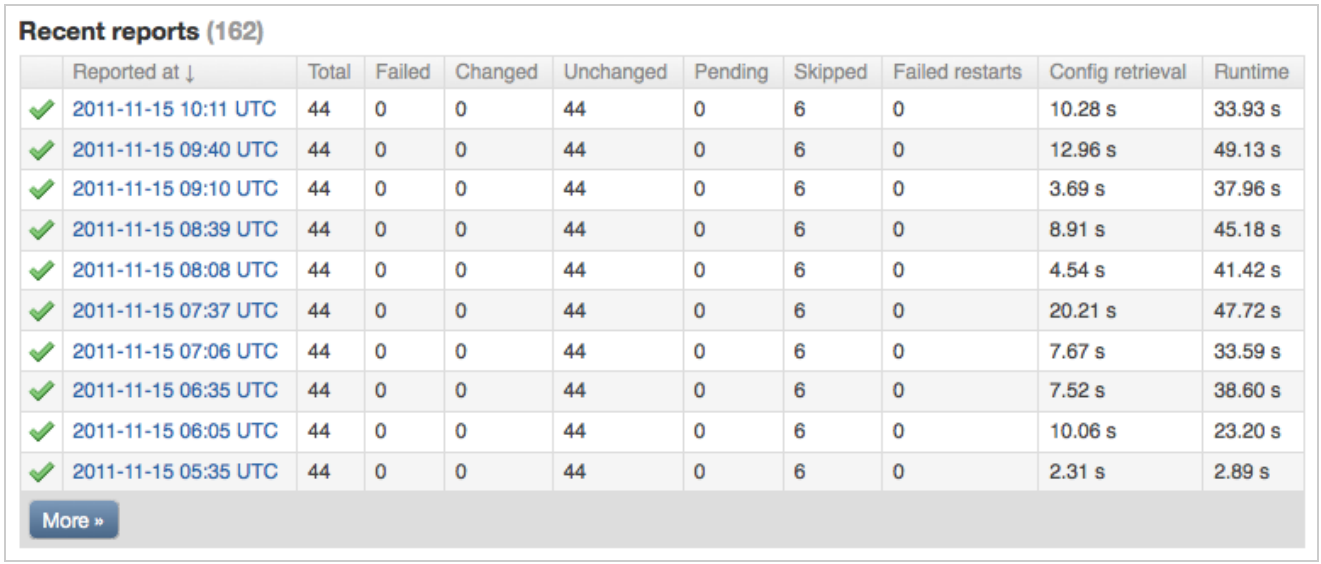
Note: Run status histograms also appear on group pages, class pages, and run status pages.

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)

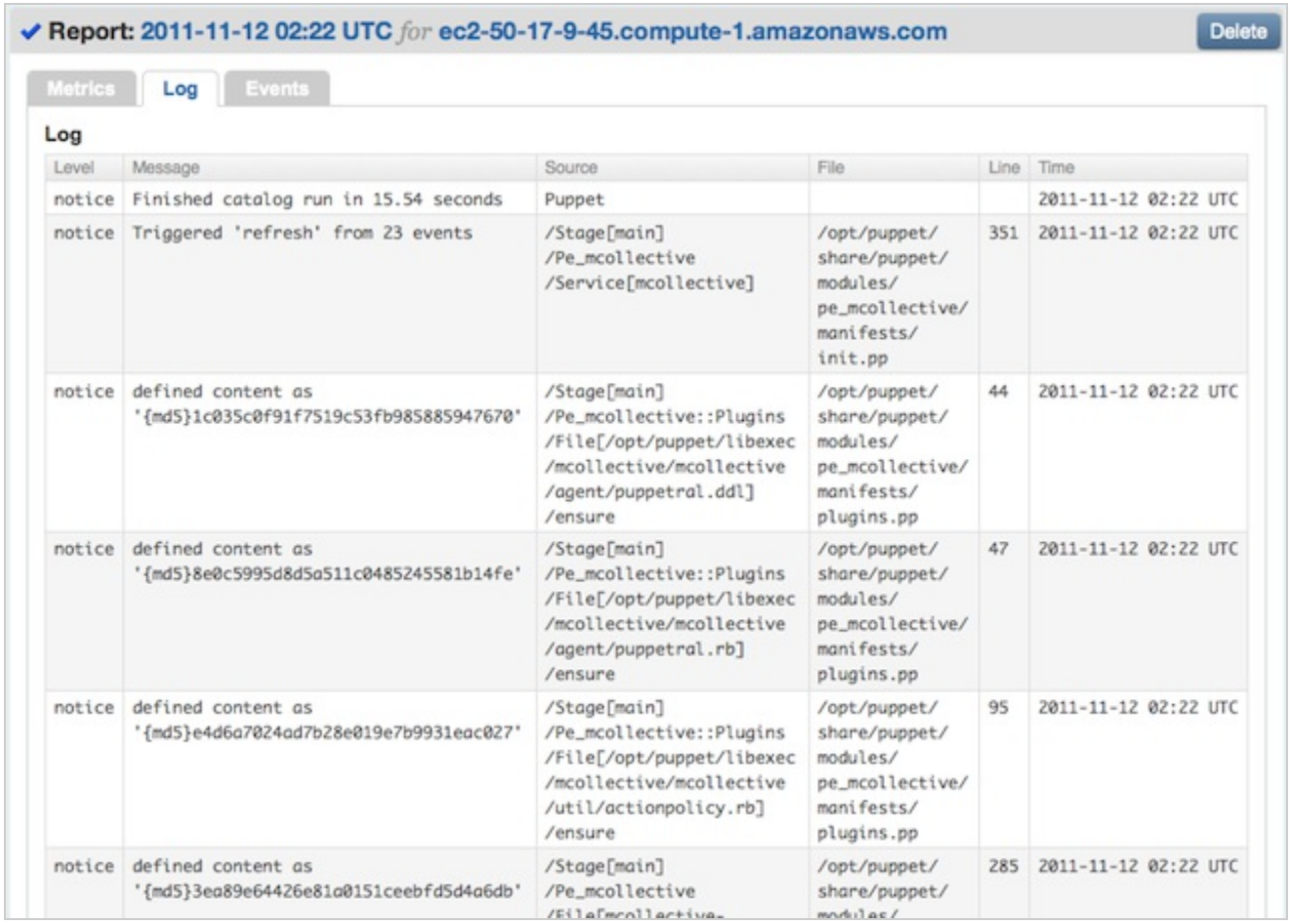
	Reported at ↓	Total	Failed	Changed	Unchanged	Pending	Skipped	Failed restarts	Config retrieval	Runtime
✓	2011-11-15 10:11 UTC	44	0	0	44	0	6	0	10.28 s	33.93 s
✓	2011-11-15 09:40 UTC	44	0	0	44	0	6	0	12.96 s	49.13 s
✓	2011-11-15 09:10 UTC	44	0	0	44	0	6	0	3.69 s	37.96 s
✓	2011-11-15 08:39 UTC	44	0	0	44	0	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 »

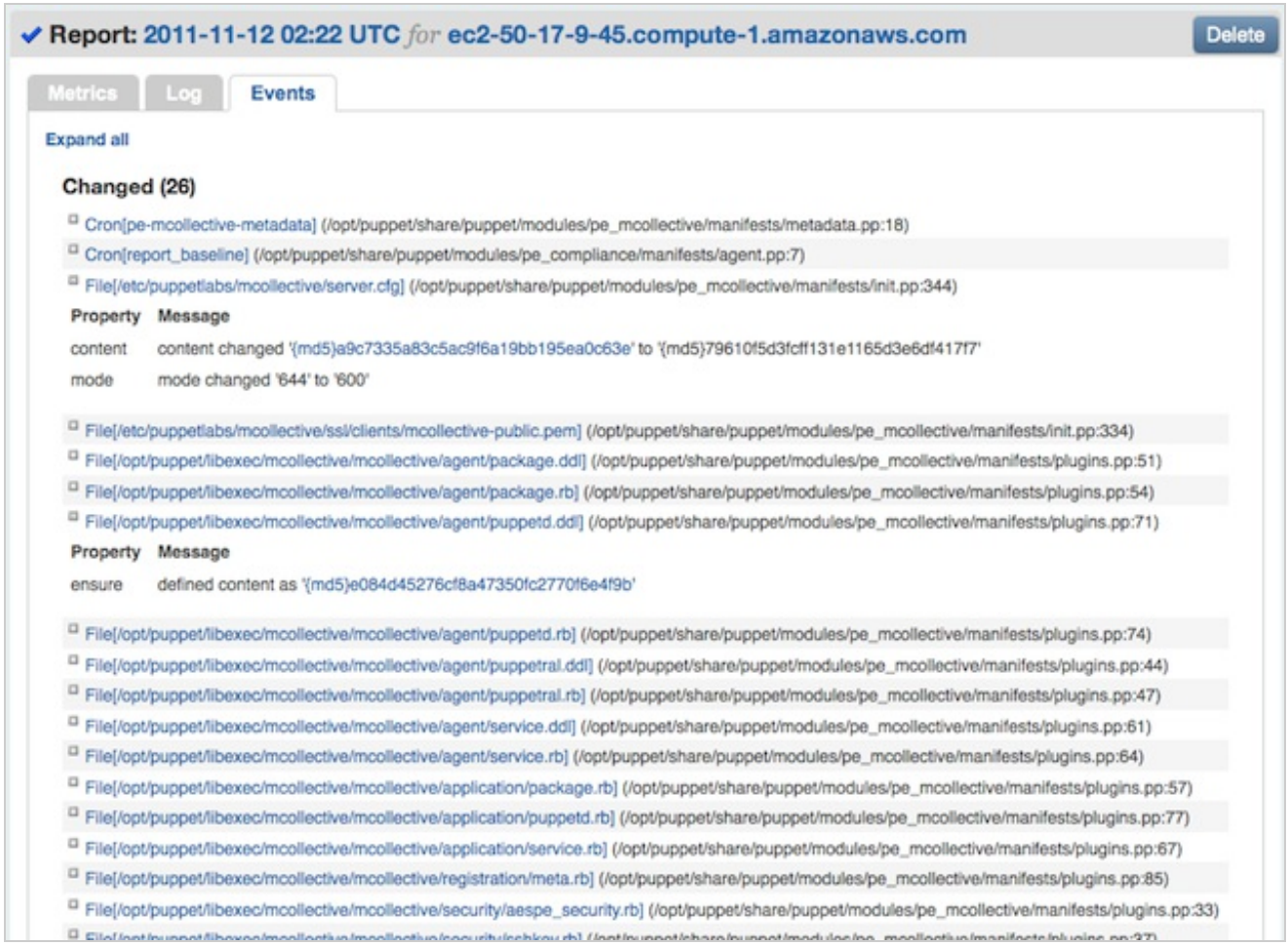
Each report represents a single Puppet run. Clicking a report will take you to a tabbed view that splits the report up into metrics, log, and events.

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.

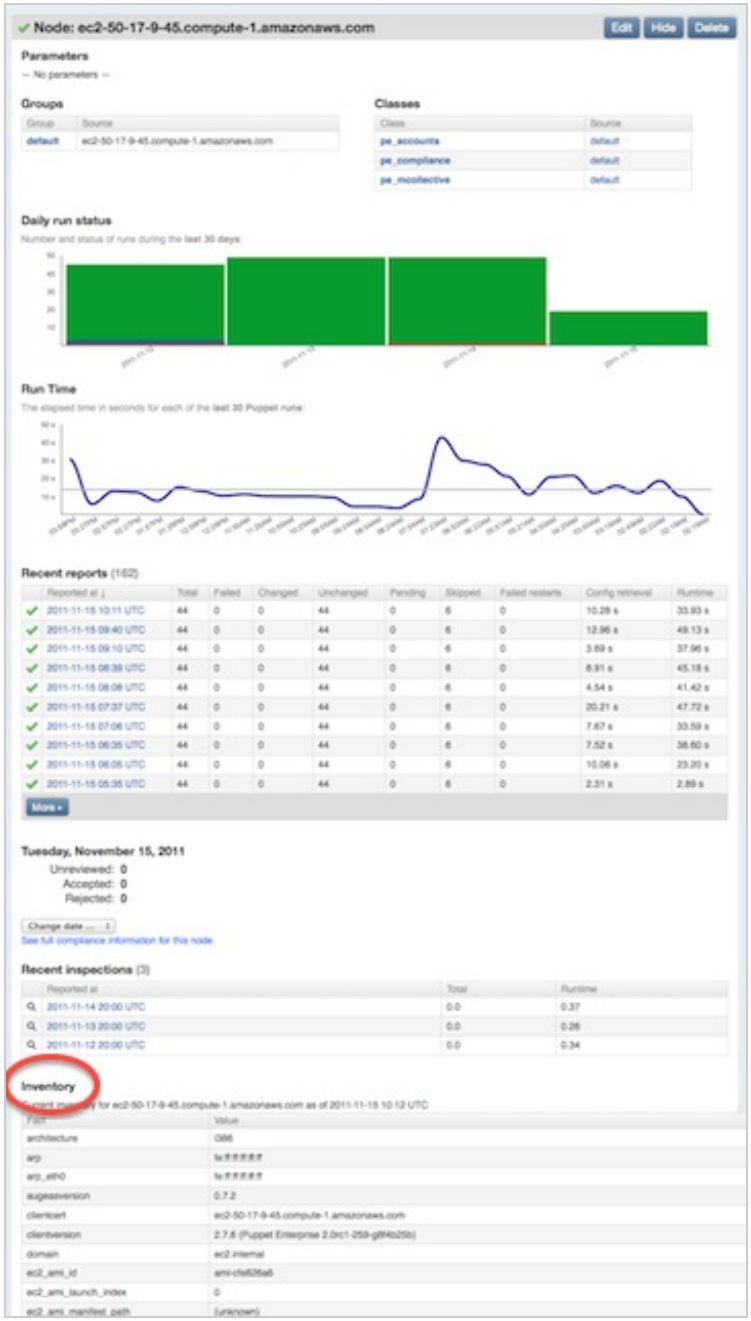


Events is a list of the resources the run managed, sorted by whether any changes were made. You can click on a changed resource to see which attributes were modified.



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)

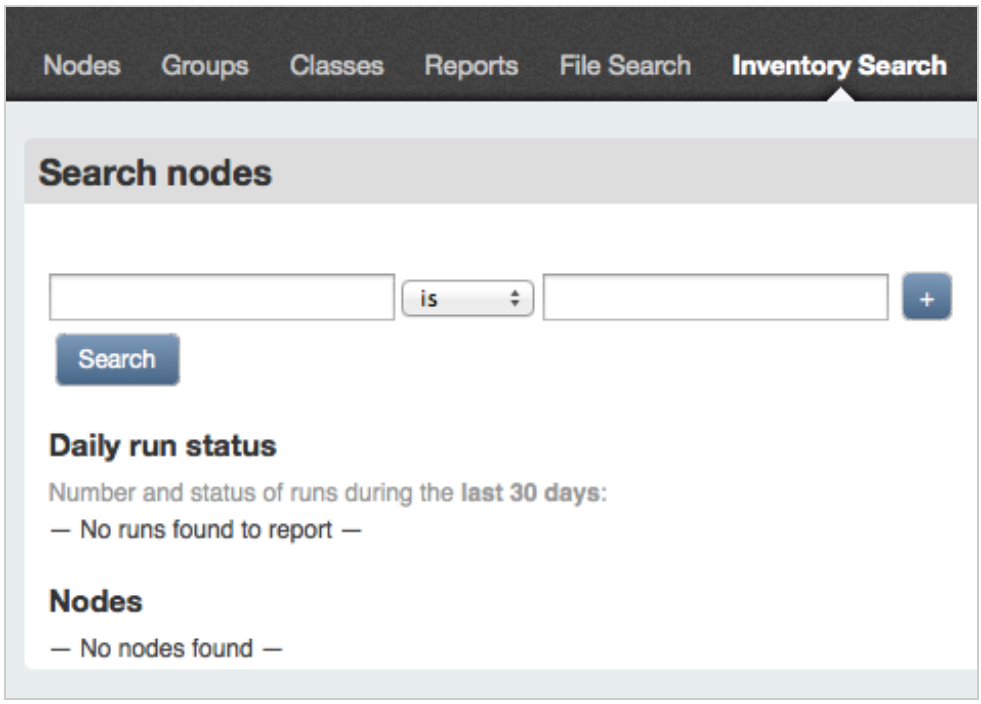
ec2_block_device_mapping_ami	/dev/sda1
ec2_block_device_mapping_ephemeral0	/dev/sda2
ec2_block_device_mapping_root	/dev/sda1
ec2_hostname	ip-10-202-214-228.ec2.internal
ec2_instance_id	i-353eb456
ec2_instance_type	t1.micro
ec2_kernel_id	aki-407d9529
ec2_local_hostname	ip-10-202-214-228.ec2.internal
ec2_local_ipv4	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

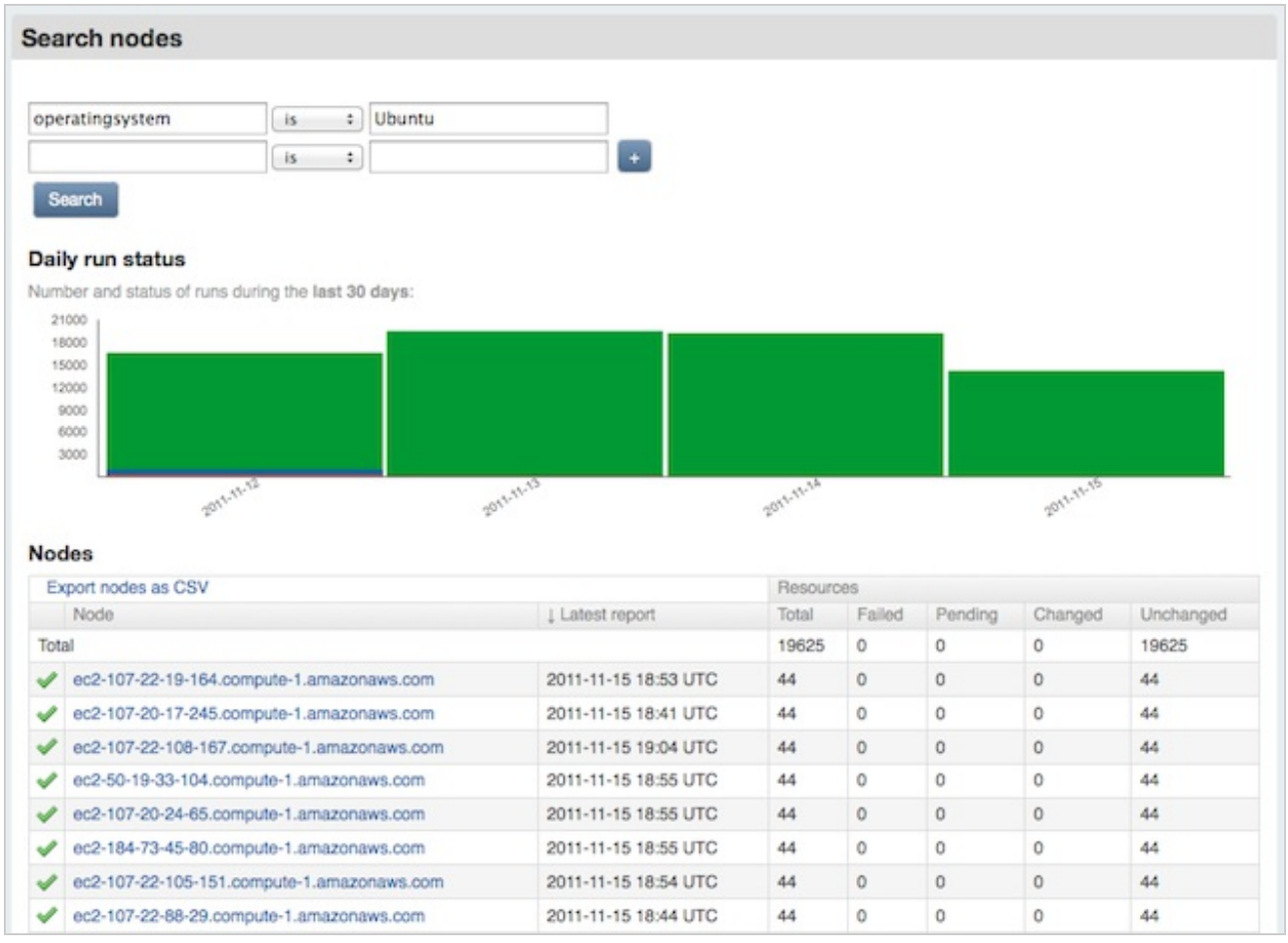
Facts include things like the operating system (`operatingsystem`), the amount of memory (`memorytotal`), and the primary IP address (`ipaddress`). You can also add arbitrary custom facts to your Puppet modules, and they too will show up in the inventory.

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 further filter the search results , and you can change the comparison criteria for each one.

- Next: Managing Users

Managing Console Users

Starting with PE 2.5, the console has supported individual user management, access and authentication. Instead of a single, shared username and password authenticated over HTTP with SSL, the console allows secure individual user accounts with different access privileges. Specifically, user accounts allow the assignment of one of three access levels: read-only, read-write, or admin. As of PE 2.6, users can also be managed using external, third-party authentication services such as LDAP, Active Directory or Google Accounts.

Following standard security practices, user passwords are hashed with a salt and then stored in a database separated from other console data. Authentication is built on CAS, an industry standard, single sign-on protocol.

Note: By default, CAS authentication for the console runs over port 443. If your console needs to access CAS on a different host/port, you can configure that in `/etc/puppetlabs/console-auth/cas_client_config_yml`.

User Access and Privileges

Depending on the access privileges assigned to them, users will be able to see and access different parts of the console:

Read-Only Users can only view information on the console, but cannot perform any actions. In particular, read-only users are restricted from:

- cloning resources in Live Management
- accessing the Control Puppet tab in Live Management
- accessing the Advanced Tasks tab in Live Management
- accepting or rejecting changes to the baseline in Compliance
- adding, editing, or removing nodes, groups, or classes

Read-Write Users have access to all parts of the console EXCEPT the user-management interface. Read-write users can interact with the console and use it to perform node management tasks.

Admin Users have unrestricted access to all parts of the console, including the user-management interface. Through this interface, admin users can:

- add a new user
- delete a user
- change a user's role
- re-enable a disabled user
- disable an enabled user
- edit a user's email

- prompt a change to the user's password

There is one exception to this: admin users cannot disable, delete or change the privileges of their own accounts. Only another admin user can make these changes.

Anonymous Users In addition to authenticated, per-user access, the console can also be configured to allow anonymous, read-only access. When so configured, the console can be viewed by anyone with a web browser who can access the site URL. For instructions on how to do this, visit the advanced configuration page.

Managing Accounts and Users Internally

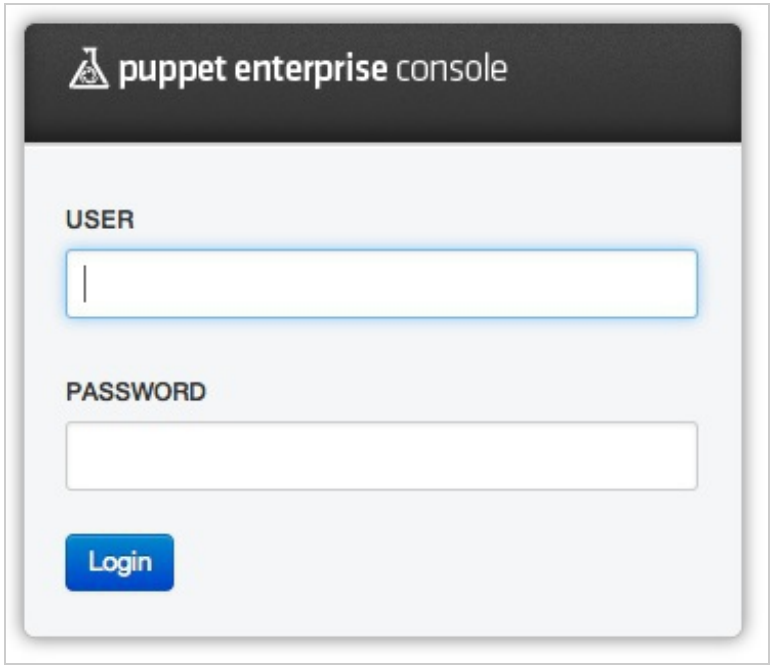
Signing Up

In order to sign up as a console user at any access level, an account must be created for you by an admin. Upon account creation, you will receive an email containing an activation link. You must follow this link in order to set your password and activate your account. The link will take you to a screen where you can enter and confirm your password, thereby completing account activation. Once you have completed activation you will be taken to the Login screen where you can enter your new credentials.

Logging In

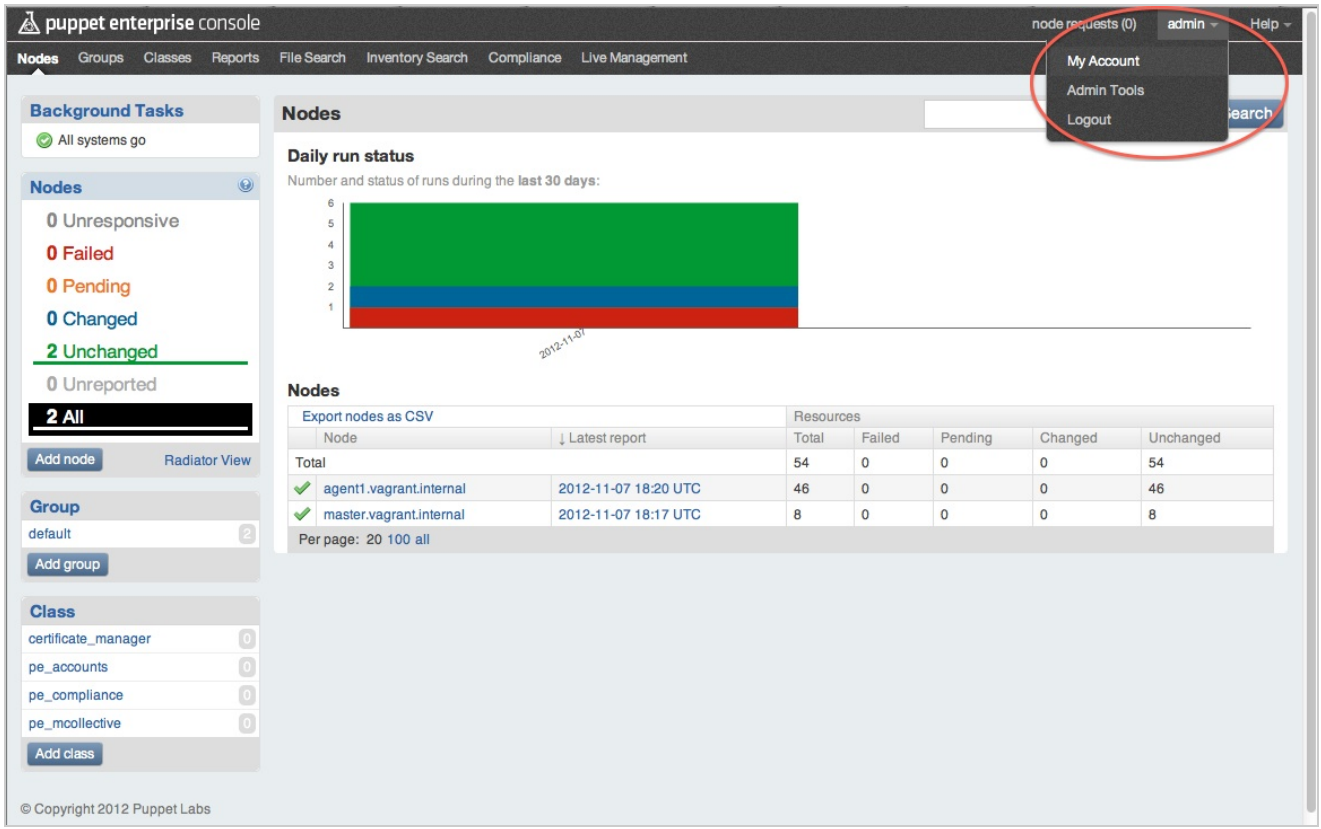
You will encounter the login screen whenever you try to access a protected part of the console. The screen will ask for your email address and password. After successfully authenticating, you will be taken to the part of the console you were trying to access.

When you're done working in the console, choose Logout from the user account menu. Note that you will be logged out automatically after 48 hours.

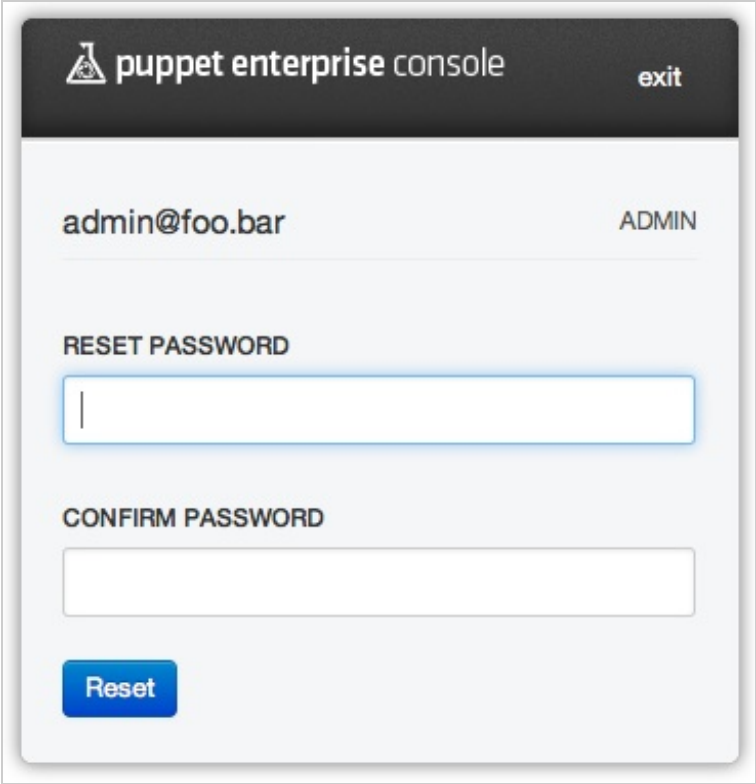


Viewing Your User Account

To view your user information, access the user account menu by clicking on your username (the first part of your email address) at the top right of the navigation bar.□

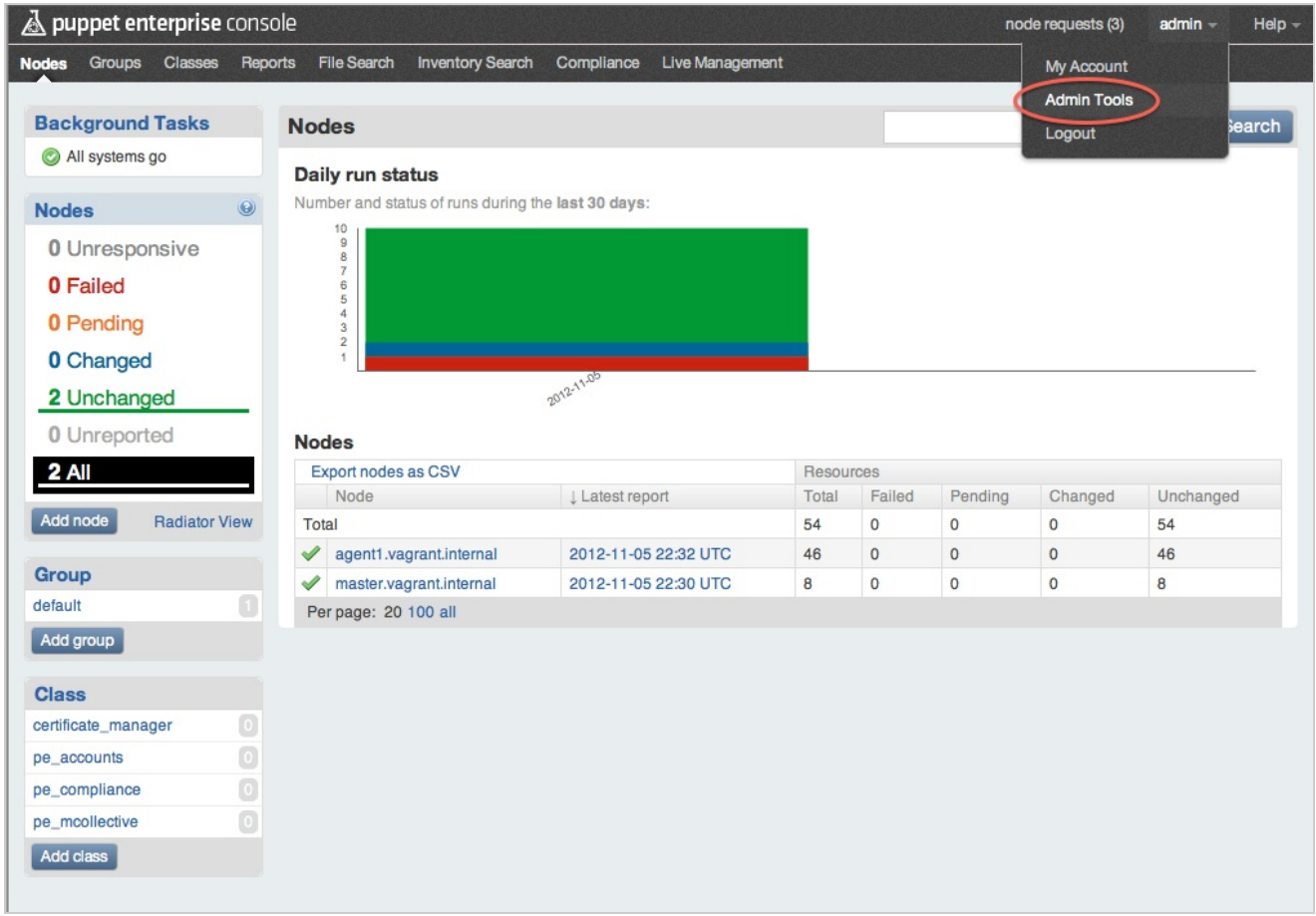


Choose My account to open a page where you can see your username/email and your user access level (admin, read-write or read-only) and text boxes for changing your password.



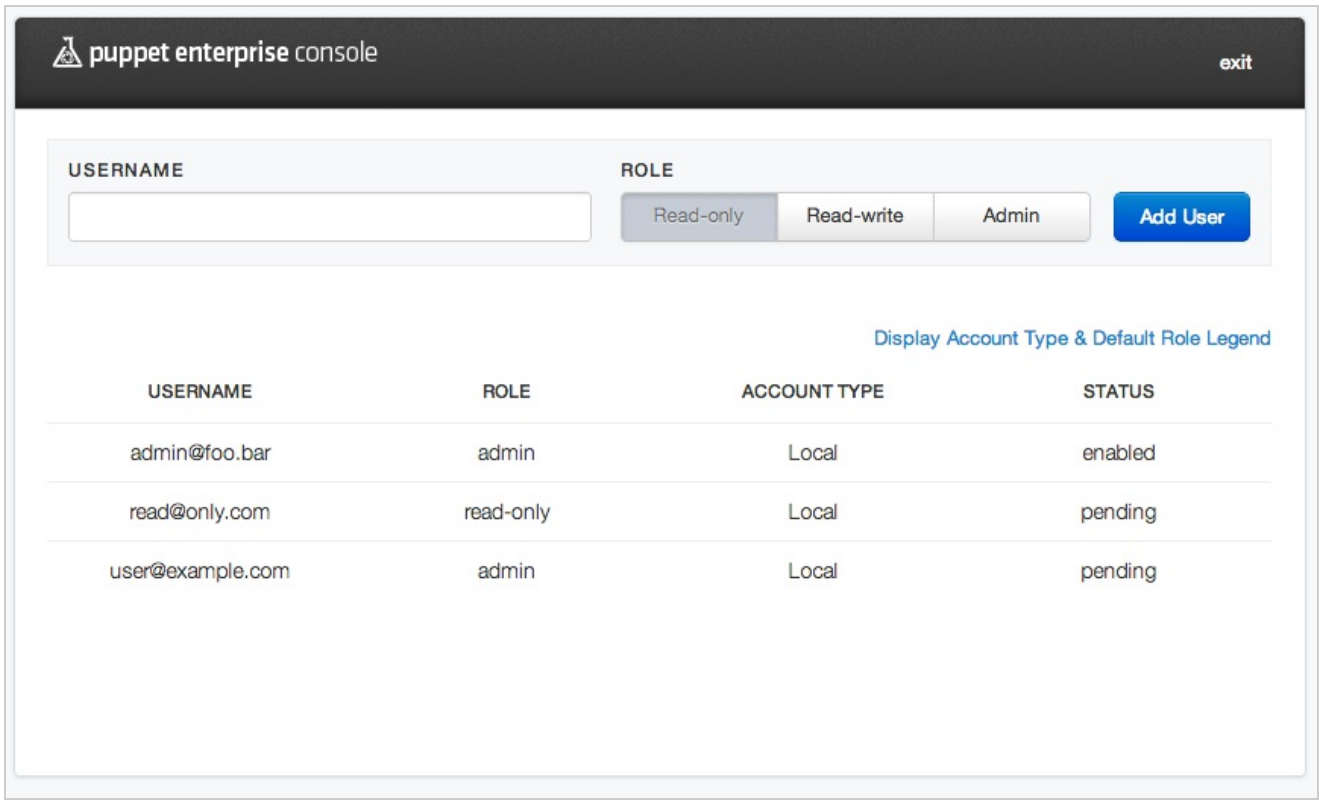
User Administration Tools

Users with admin level access can view information about users and manage their access, including adding and deleting users as needed. Admin level users will see an additional menu choice in the user account menu: Admin Tools. Users with read-write or read-only accounts will NOT see the Admin Tools menu item.

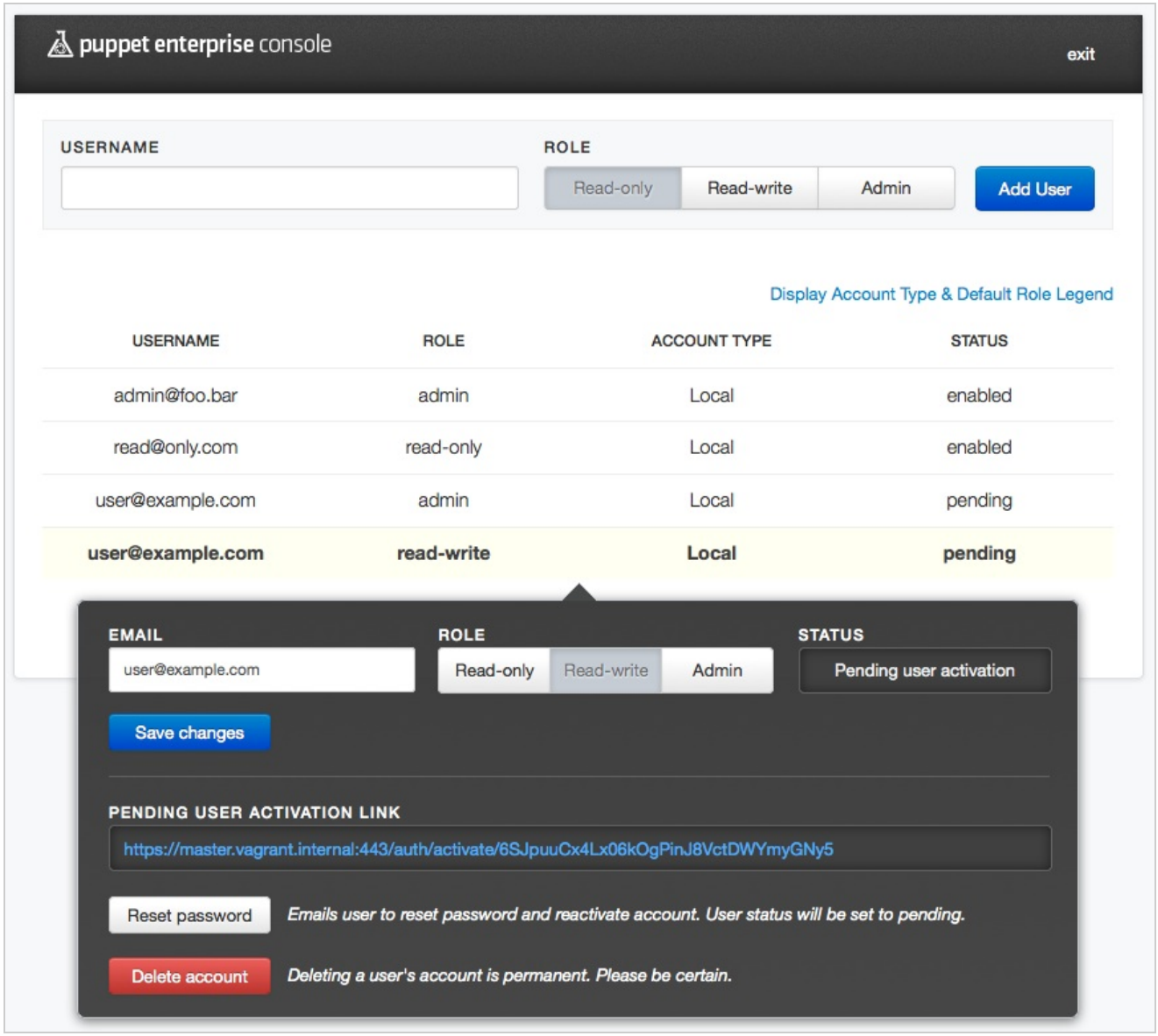


VIEWING USERS AND SETTINGS

Selecting Admin Tools will open a screen showing a list of users by email address, their access role and status. Note that users who have not yet activated their accounts by responding to the activation email and setting a password will show a status of pending.



Click on a user's row to open a pop-up pane with information about that user. The pop-up will show the user's name/email, their current role, their status and other information. If the user has not yet validated their account, you will also see the link that was generated and included in the validation email. Note that if there is an SMTP issue and the email fails to send, you can manually send this link to the user.



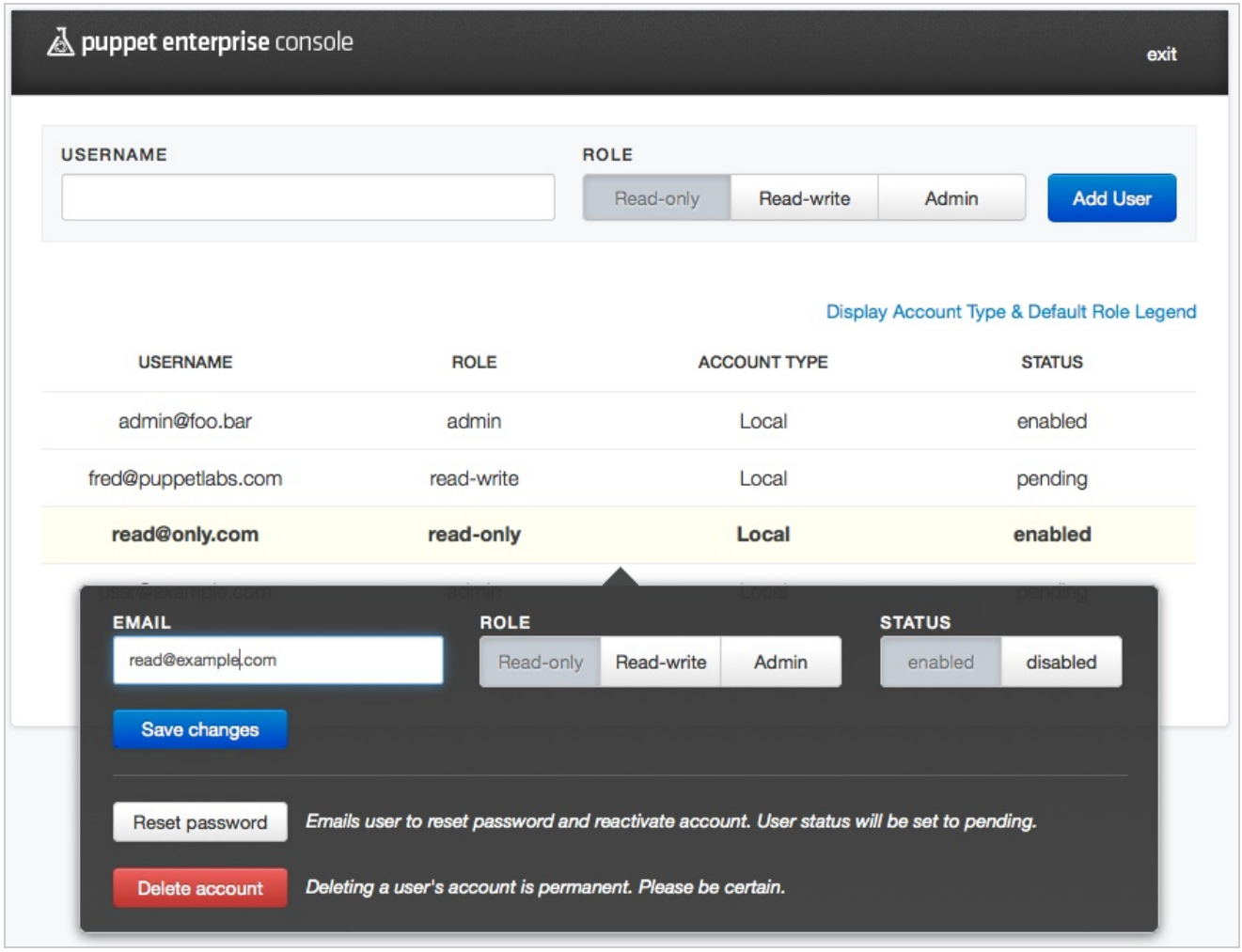
MODIFYING USER SETTINGS

To modify the settings for a given user, click on the user's row to open the pop-up pane. In this pane, you can change their role and their email address or reset their password. Don't forget to click the Save changes button after making your edits.

Note that resetting a password or changing an email address will change that user's status back to Pending, which will send them another validation email and require them to complete the validation and password setting process again.

For users who have completed the validation process, you can also enable or disable a user's account. Disabling the account will prevent that user from accessing the console, but will not

remove them from the users database.



ADDING/DELETING USERS

To add a new user, open the user admin screen by choosing Admin Tools in the user menu. Enter the user's email address and their desired role, then click the "Add user" button. The user will be added to the list with a pending status and an activation email will be automatically sent to them.

To delete an existing user (including pending users), click on the user's name in the list and then click the Delete account button. Note that deleting a user cannot be undone, so be sure this is what you want to do before proceeding.

Creating Users From the Command Line

New console users can also be created from the command line. This can be used to automate user creation or to import large numbers of users from an external source at once.

On the console server, the following command will add a new user:

```
$ sudo /opt/puppet/bin/rake -f /opt/puppet/share/console-auth/Rakefile
db:create_user USERNAME="<email address>" PASSWORD="<password>" ROLE="< Admin |
Read-Only | Read-Write >"
```

> Note: The “EMAIL” variable used in PE 2.5 has been changed to “USERNAME” in PE 2.7.

Thus, to add a read-write user named jones@example.com, you would run:

```
$ sudo /opt/puppet/bin/rake -f /opt/puppet/share/console-auth/Rakefile
db:create_user USERNAME="jones@example.com" PASSWORD="good_password_1"
ROLE="Read-Write"
```

You cannot currently delete or disable users or reset passwords from the command line.

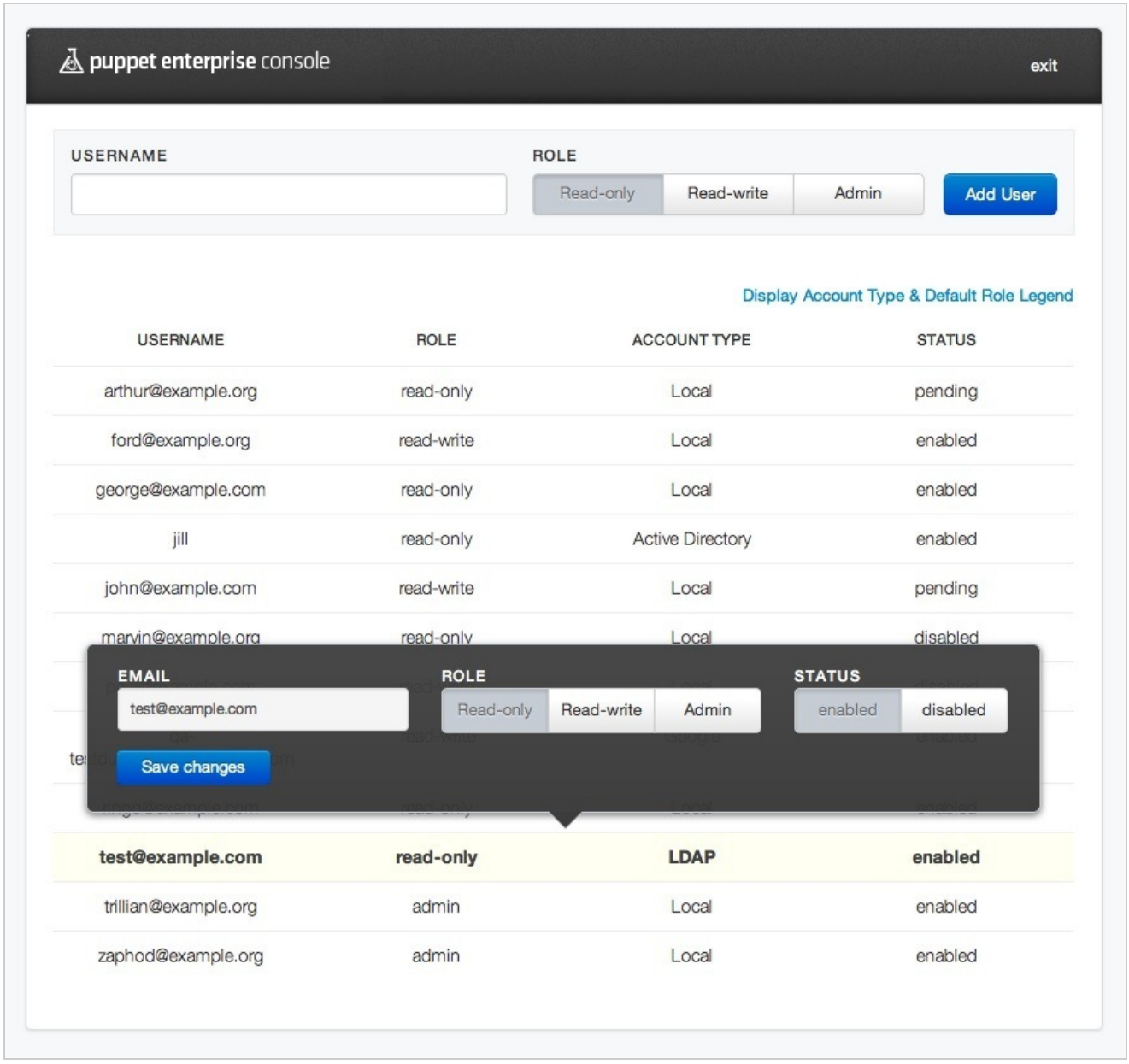
Using Third-Party Authentication Services

As of PE 2.6, admins can use external, third-party authentication services to manage user access. The following external services are supported:

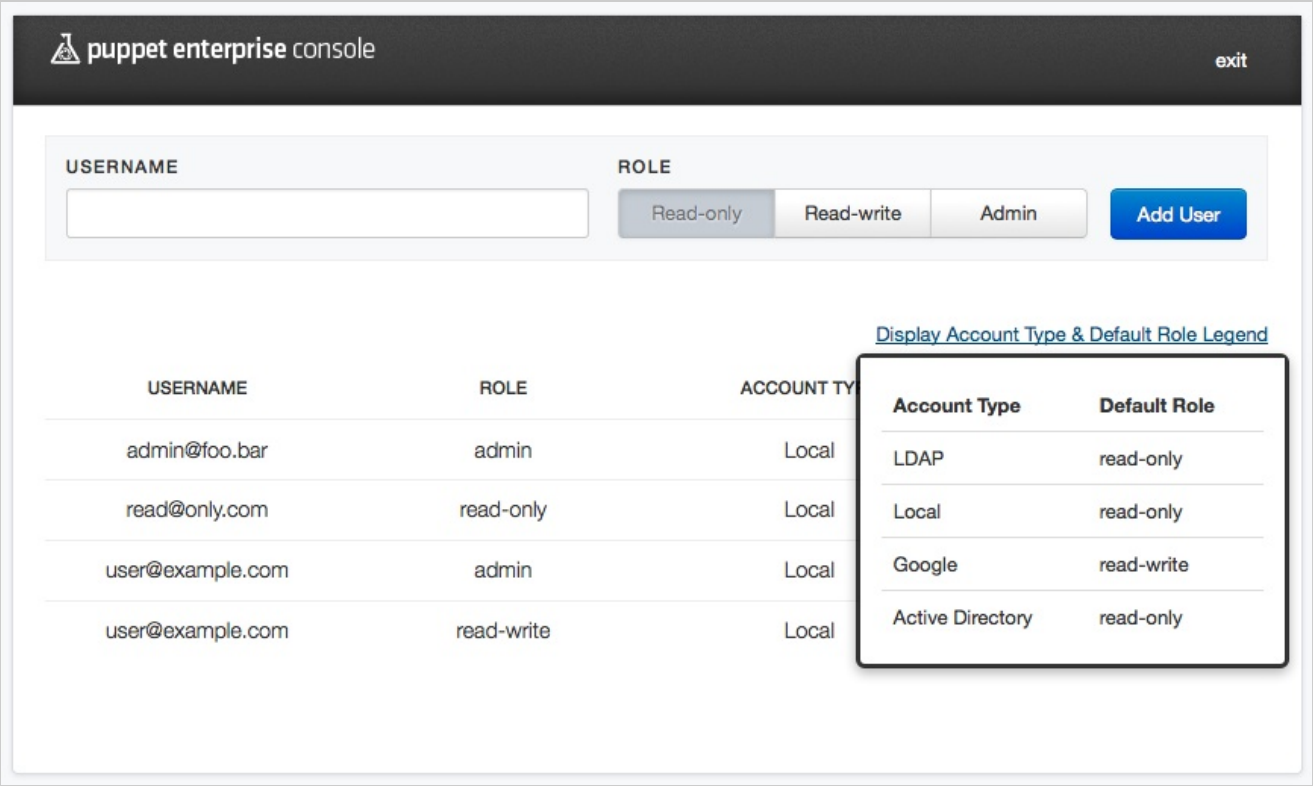
- LDAP
- Active Directory
- Google accounts

When using third-party services, the console’s RBAC retains control over the access privileges. When a user logs in using an external service, the console will check their level of access privileges. If they have never logged in before, they are assigned a default role. (This role can be configured. See “Configuration” below.) External users’ access privileges are managed in the same manner as internal users, via the console’s user administration interface.

The account interface for an externally authenticated user differs slightly from internal users in that external users do not have UI for changing their passwords or deleting accounts.



Admins will also notice additional UI on the user administration page which indicates the authentication service ("Account Type") being used for a given user and a link to a legend that lists the external authentication services and the default access privileges given to users of a given service.



Configuration

To use external authentication, the following two files must be correctly configured:

1. `/etc/puppetlabs/console-auth/cas_client_config.yml`
2. `/etc/puppetlabs/rubycas-server/config.yml`

> Note: If you upgraded from PE 2.5, your `cas_client_config.yml` and `rubycas-server/config.yml` files will not have the relevant commented-out sections, as they were added for 2.6 and the config files are not overwritten by the upgrader.
>
> To find example config code that can be copied and pasted into the live config files, look in files with the same names and either the `.rpmnew` or `.dpkg-new` extension.

CONFIGURING `CAS_CLIENT_CONFIG.YML`

The `cas_client_config.yml` file contains several commented-out lines under the `authorization:` key. Simply un-comment the lines that correspond to the RubyCAS authenticators you wish to use. You can also set the default access level for a given authentication service using `default_role`, which accepts the following values: `read-only`, `read-write`, or `admin`.

Each entry consists of the following:

- A common identifier (e.g. `local`, or `ldap`, etc.) which is used in the console_auth database and corresponds to the classname of the RubyCAS authenticator.
- `default_role` which defines the role to assign to users by default

- `description` which is simply a human readable description of the service

  authorization: local: default_role: read-only description: Local # ldap: # default_role: read-only # description: LDAP # activedirectoryldap: # default_role: read-only # description: Active Directory # google: # default_role: read-write # description: Google

CONFIGURING `RUBYCAS-SERVER/CONFIG.YML`

This file is used to configure RubyCAS to use external authentication services. As before, you will need to un-comment the classes for the third-party services you wish to enable. The values for the listed keys are LDAP and ActiveDirectory standards. If you are not the administrator of those databases, you should check with that administrator for the correct values.

The authenticators are listed in the file in following manner:

```
- class: CASServer::Authenticators::Google
    restricted_domain: example.com

- class: CASServer::Authenticators::LDAP
    ldap:
        host: tb-driver.example.com
        port: 389
        base: dc=example,dc=test
        filter: (objectClass=person)
        username_attribute: mail

 - class: CASServer::Authenticators::ActiveDirectoryLDAP
    ldap:
        host: winbox.example.com
        port: 389
        base: dc=example,dc=dev
        filter: (memberOf=CN=Example Users,CN=Users,DC=example,DC=dev)
        auth_user: cn=Test I. Am,cn=users,dc=example,dc=dev
        auth_password: P4ssword
```

- Next: Grouping and Classifying Nodes

Grouping and Classifying Nodes

Groups, classes, and parameters are used to control which Puppet configurations your nodes receive.

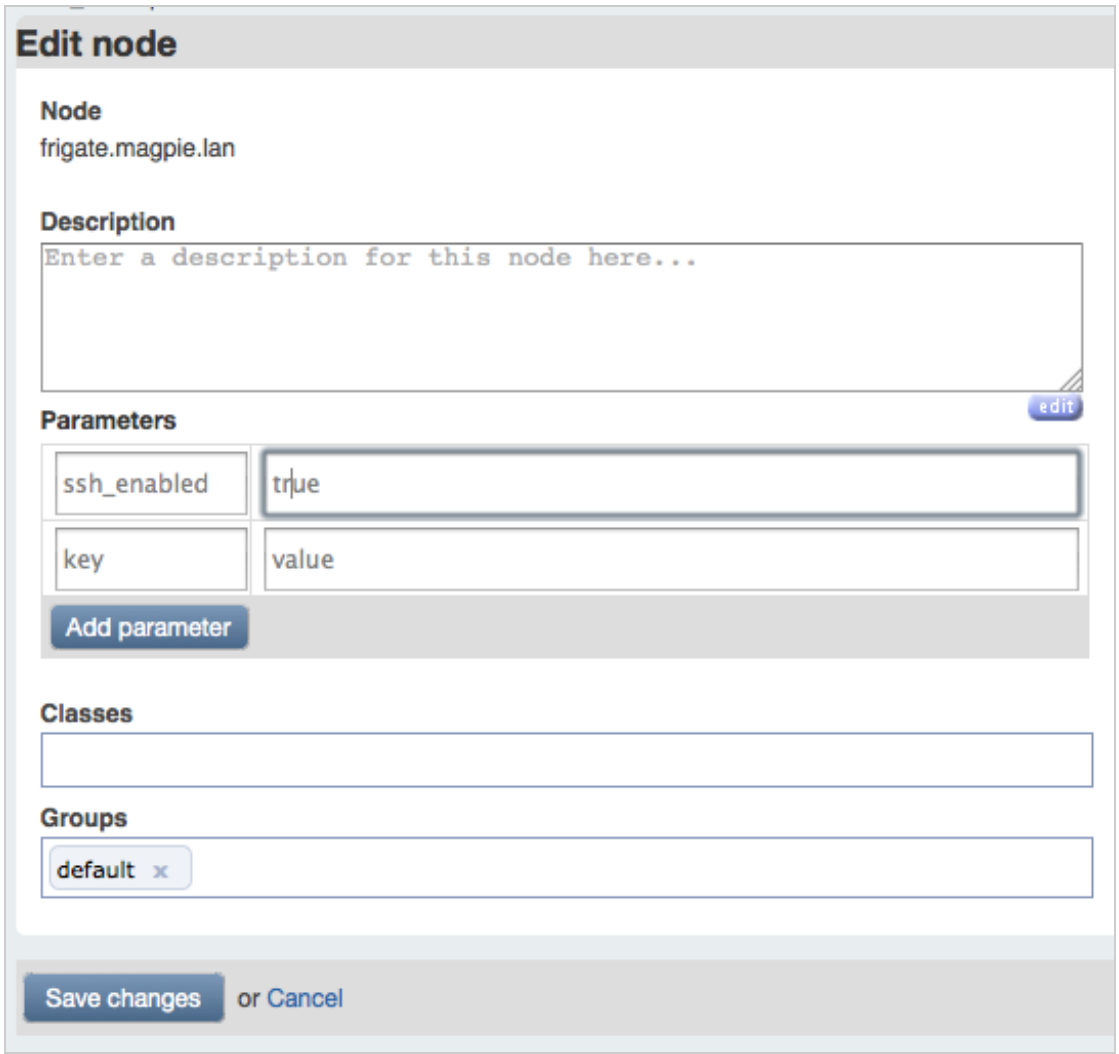
NOTE To use the console to control node configuration, you must be logged in as a read-write or admin level user. Read-only users can only view node configuration data, they cannot modify it.

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.

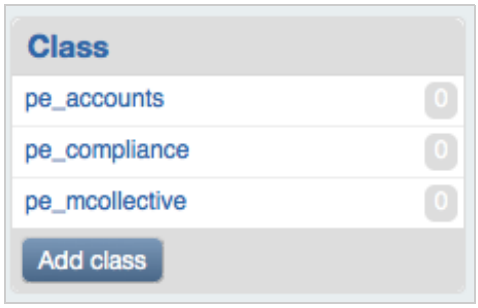
Classes

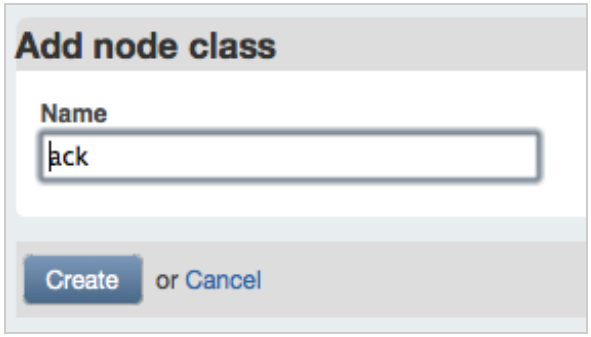
The classes the console knows about are a subset of the classes in your puppet master’s collection of modules. You must add classes to the console manually if you want to assign them to any nodes or groups.

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."



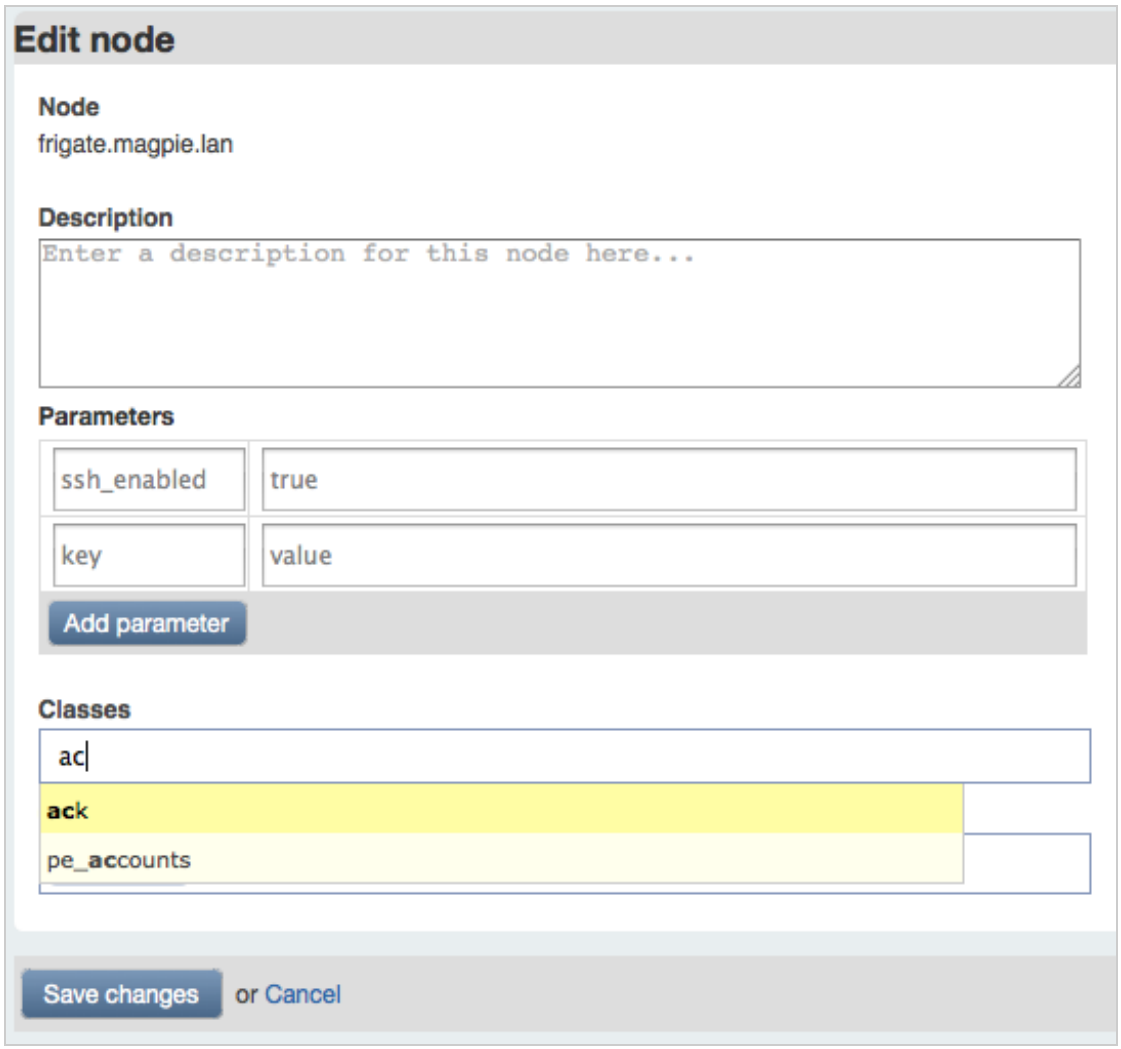


In this case, we're adding a tiny class that makes sure ack is present in `/usr/local/bin`.

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; which one you choose depends 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.

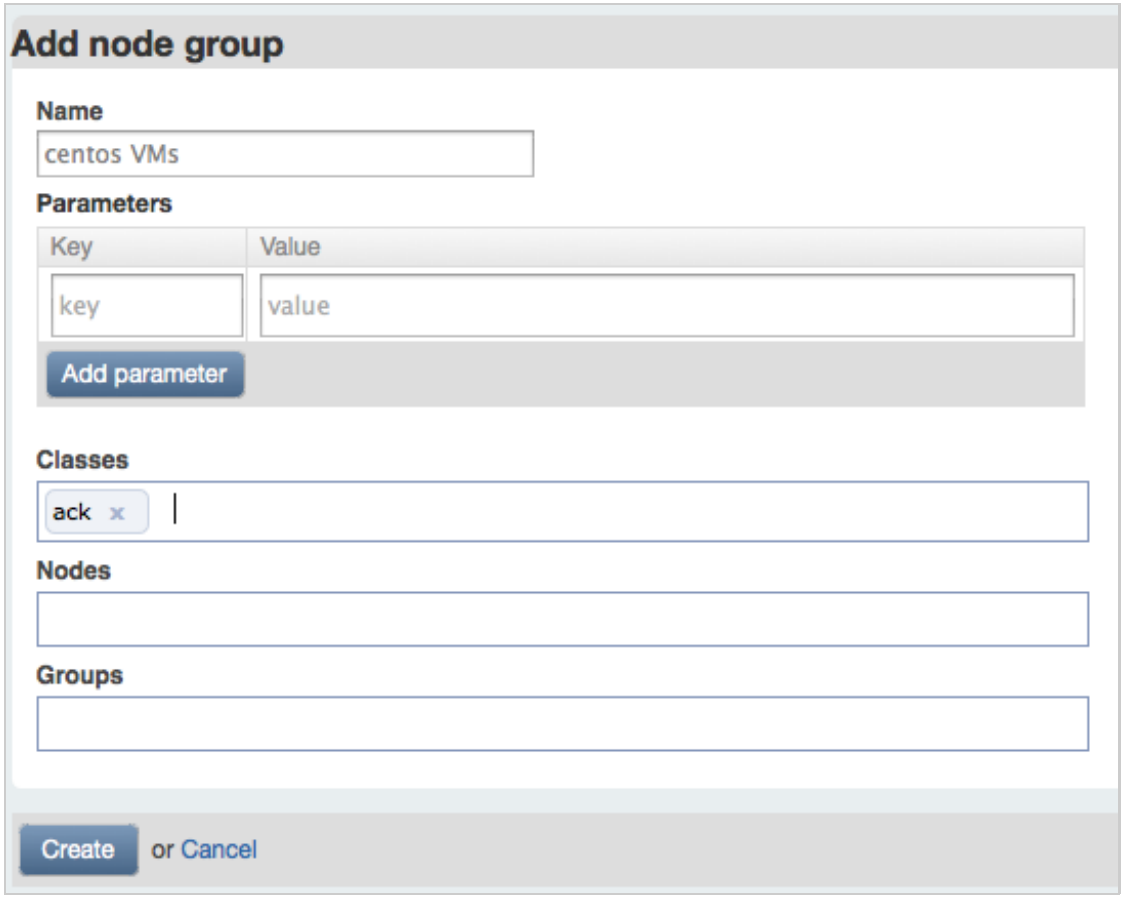
Grouping Nodes

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

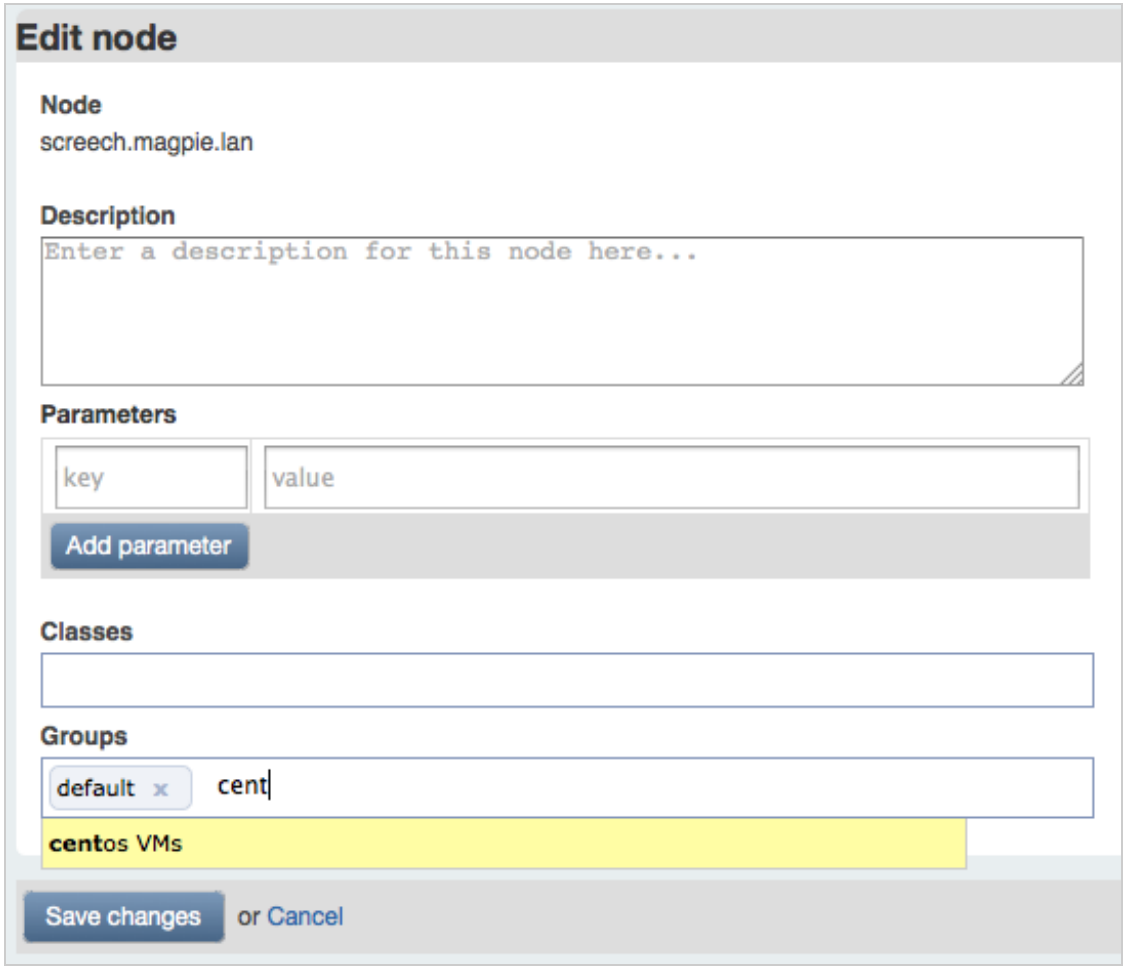
Use the “add group” button in the console’s sidebar, then enter the group’s name and any classes or parameters you want to assign.

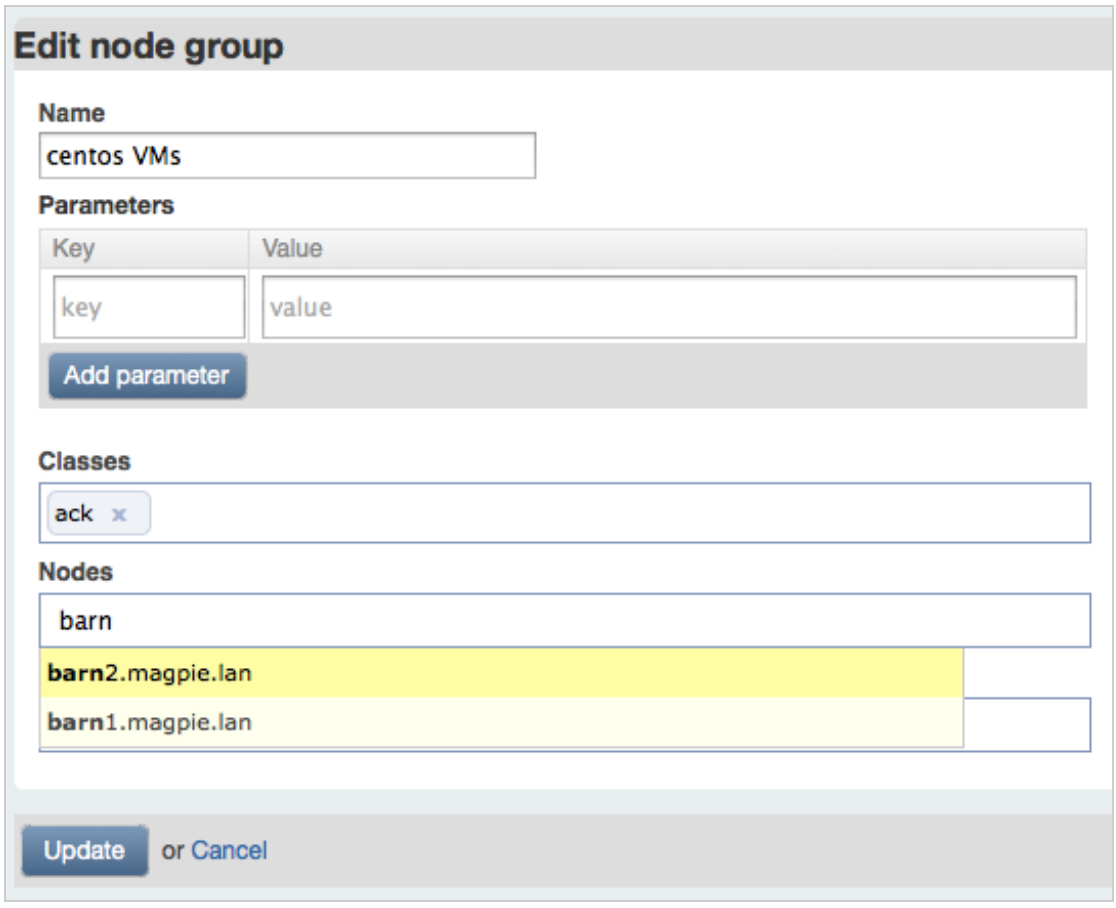


Adding Nodes to a Group

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.

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 tasks 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.

- Next: Live Management

Live Management

What is Live Management?

NOTE: To use live management, you must be logged in as a read-write or admin level user. Read-only users cannot access the live management page and will not see the associated UI elements.

Live management is a part of the console that provides 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.

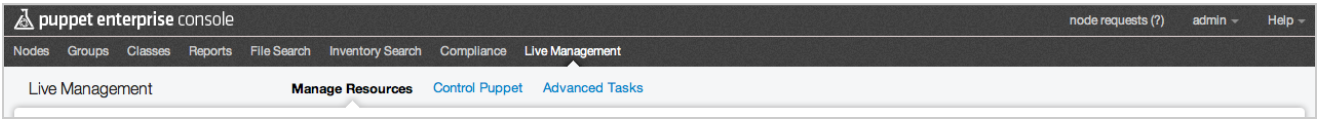
 NOTE: Live management and MCollective are not yet supported on Windows nodes.

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.

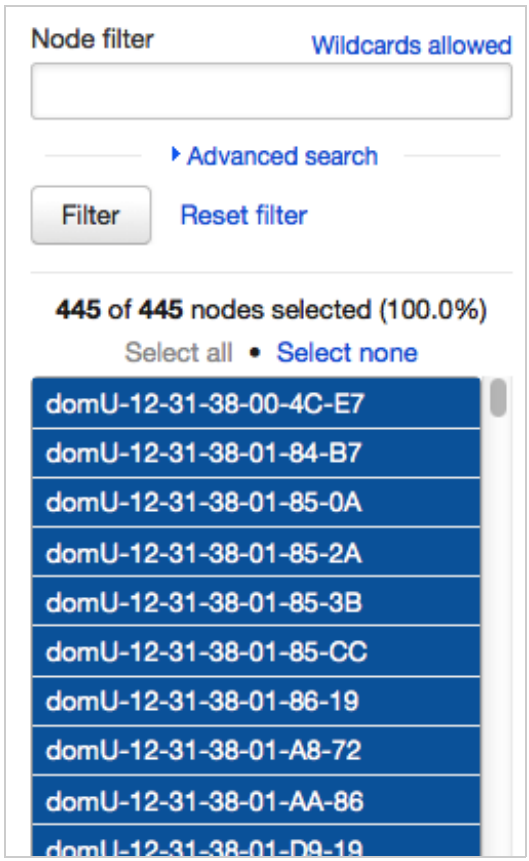


- The manage resources tab lets you browse the resources on your nodes and clone any of them across your infrastructure.
- The control Puppet tab lets you tell any node to immediately pull and apply its configuration. You can also temporarily disable puppet agent on some of your nodes to control the 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.

Subsequent 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 always the case, especially in 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.

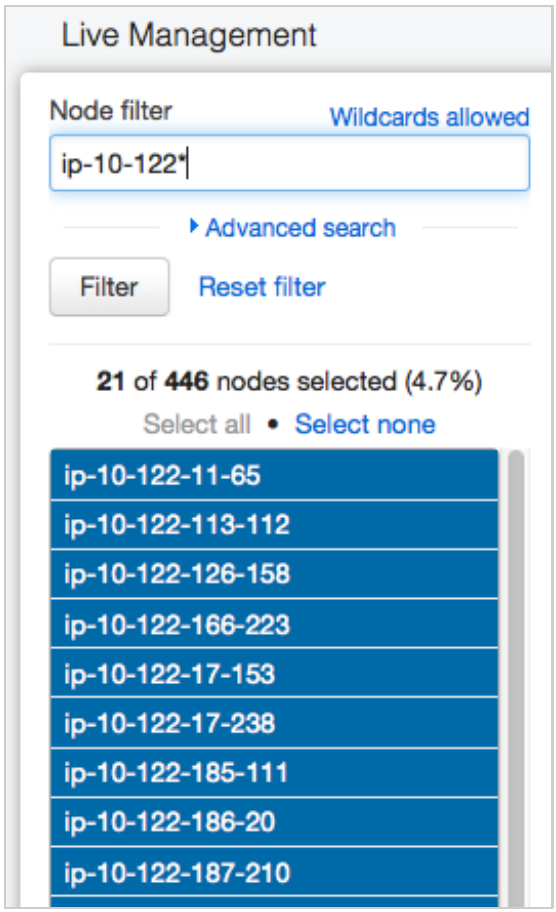
Selecting Nodes

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.



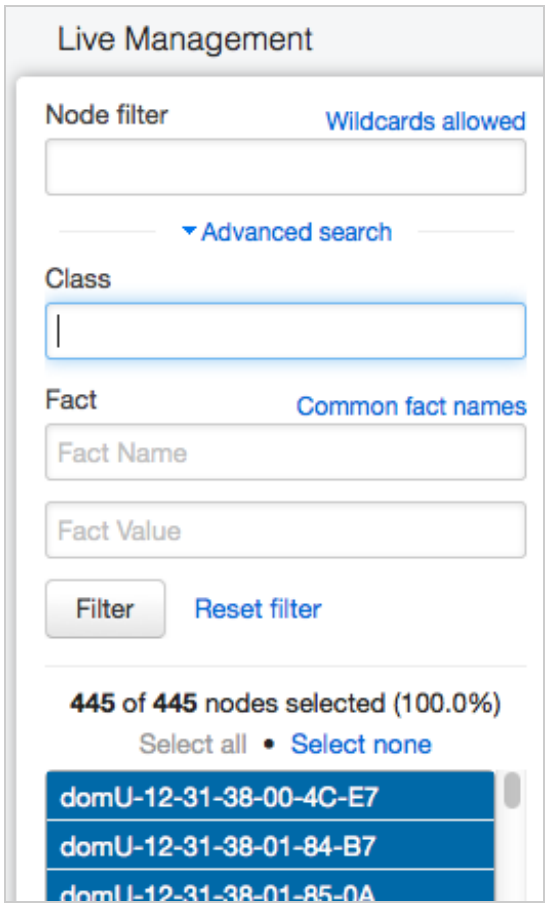
You can use the following wildcards in the node filter field:

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

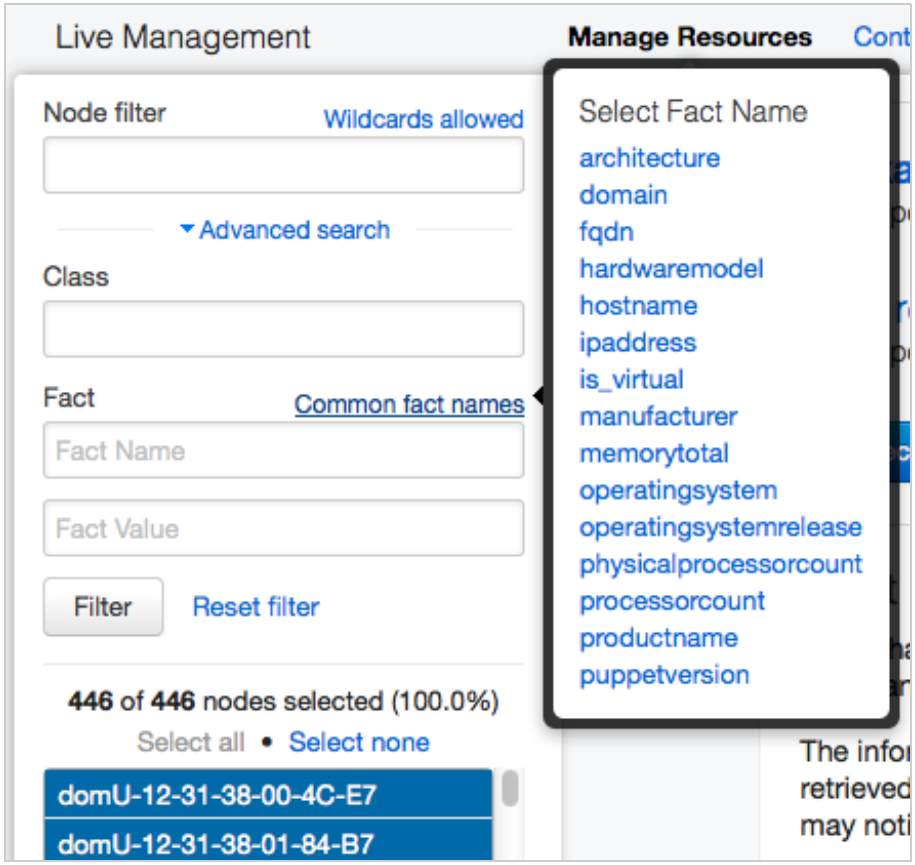
Use the “filter” button or the enter key to confirm your search, then wait for the node list to be updated.

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.

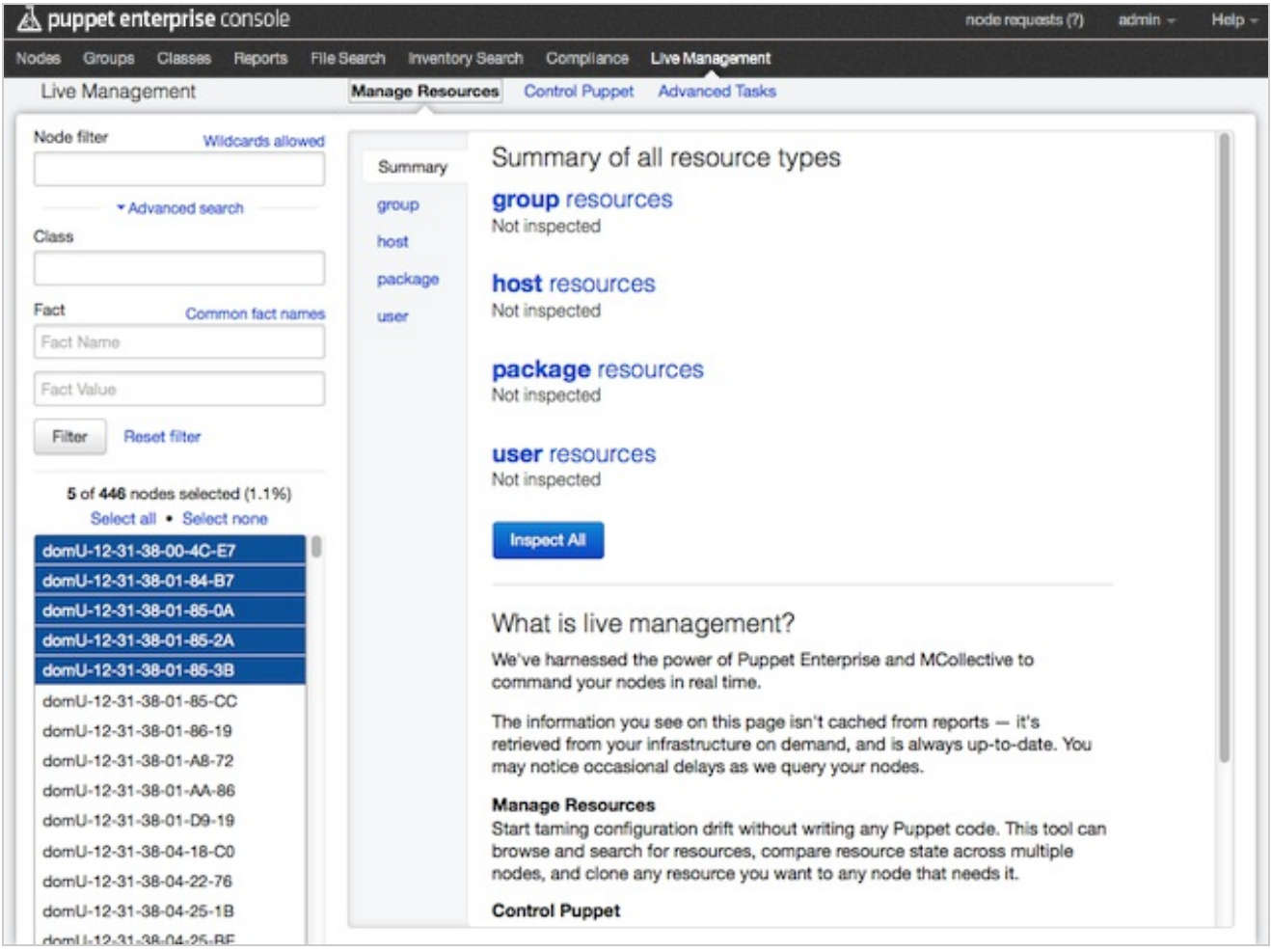
Filtering by Puppet class can be the most powerful filtering tool on this page, but it requires you to have already sorted your nodes by assigning distinctive classes to them. See the chapter on grouping and classifying nodes for more details.

- Next: Live Management: Managing Resources

Live Management: Managing Resources

NOTE: Live management and MCollective are not yet supported on Windows nodes.

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 “Inspect All” button 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.

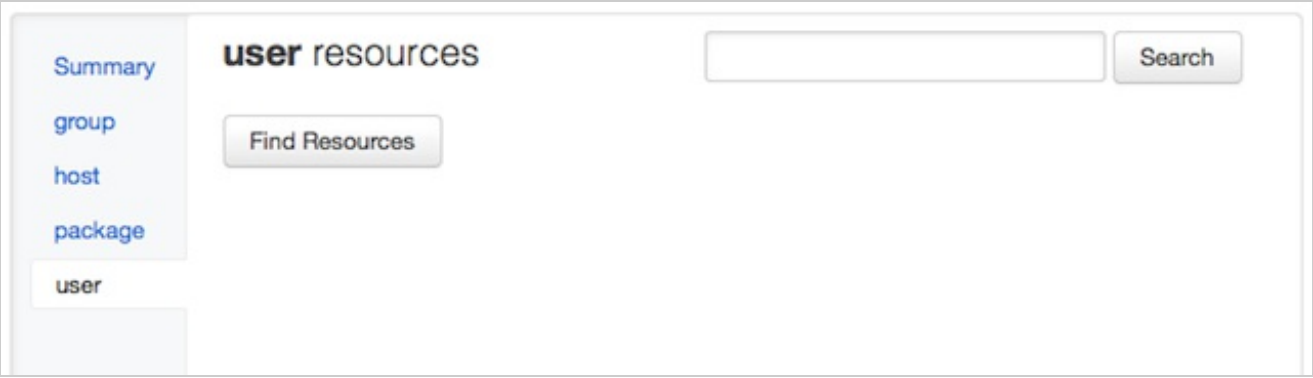
Finding Resources

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.

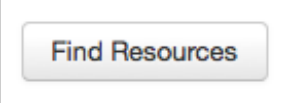
Searching and browsing only use the current selection of nodes.

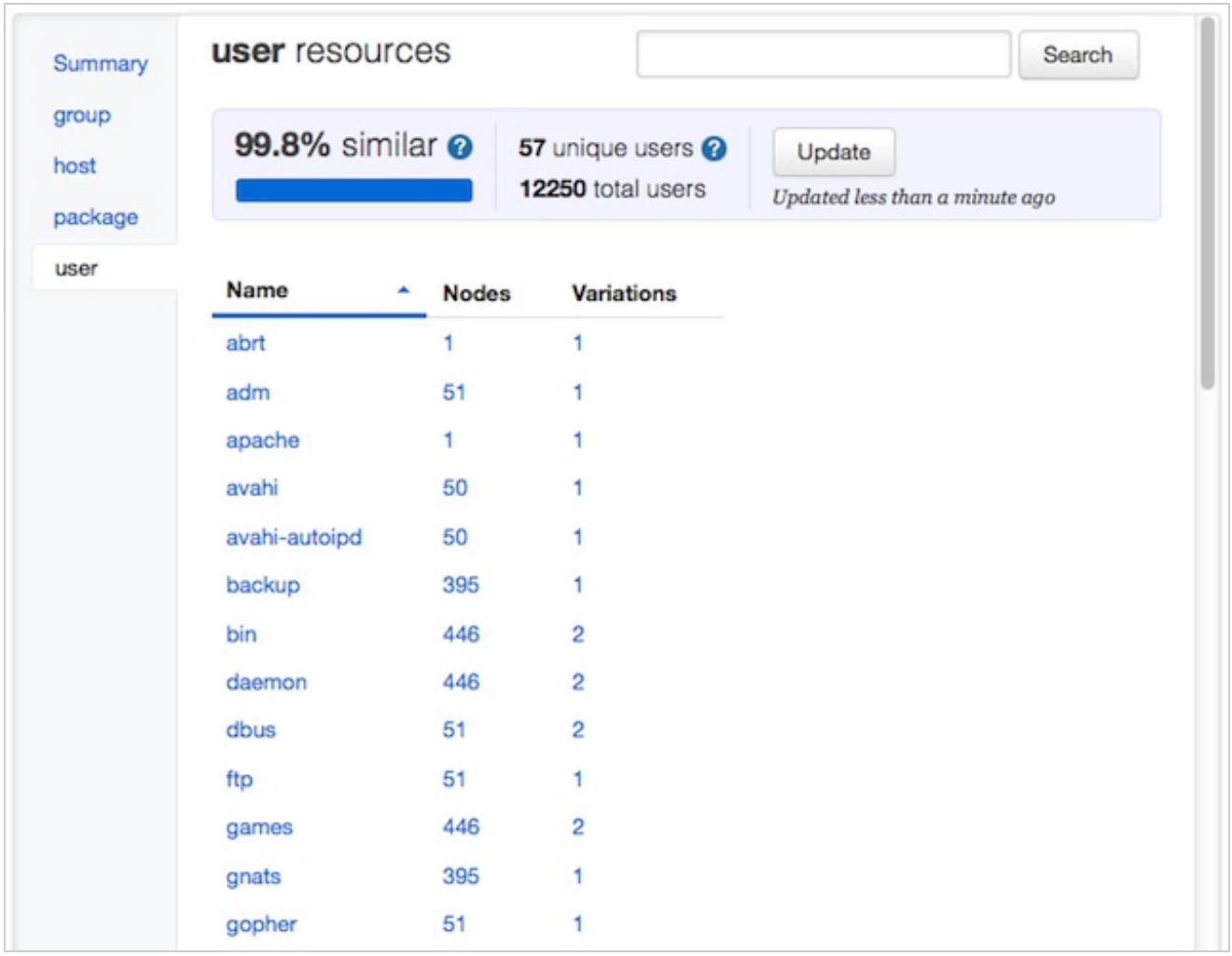
Browsing a Type

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



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

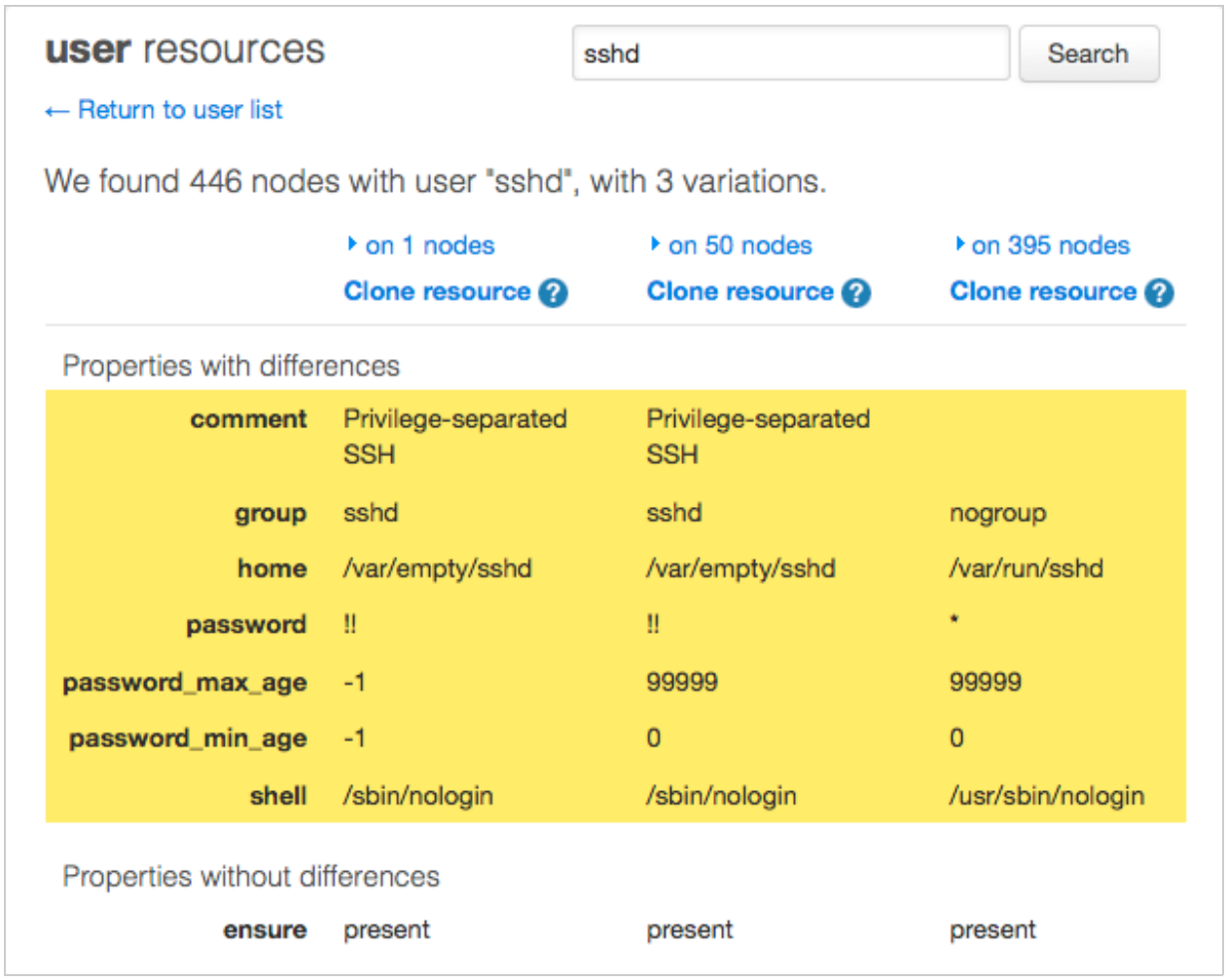




This will return a list of all resources of the selected type on the selected nodes, plus a summary of how similar the resources are. In general, a set of nodes that perform similar tasks should have very similar resources. You can make a set of nodes more similar by cloning resources across them.

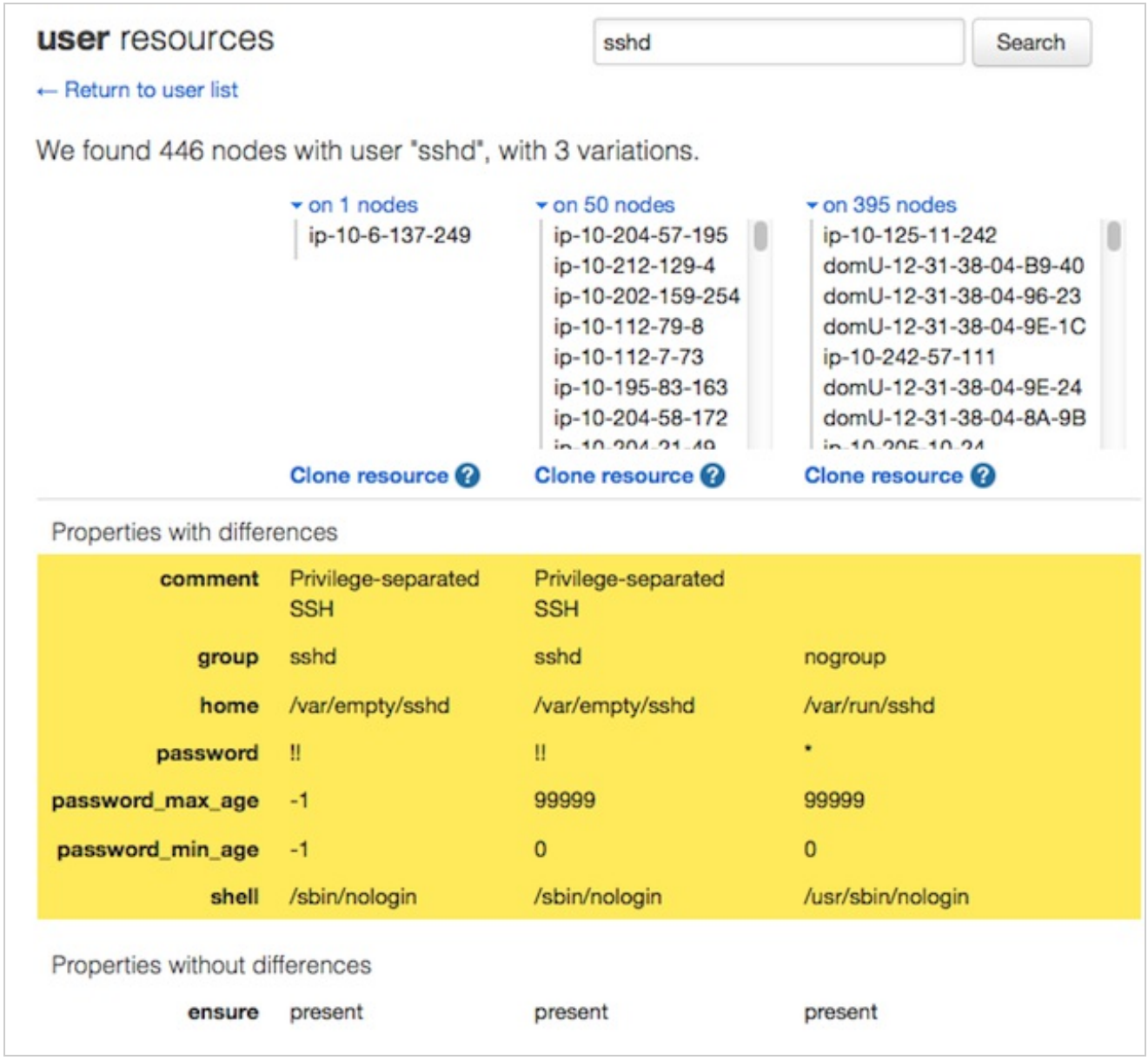
The resource list shows the name of each resource, the number of nodes it was found on, and how many variants of it were found. You can sort the list by any of these properties.

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.

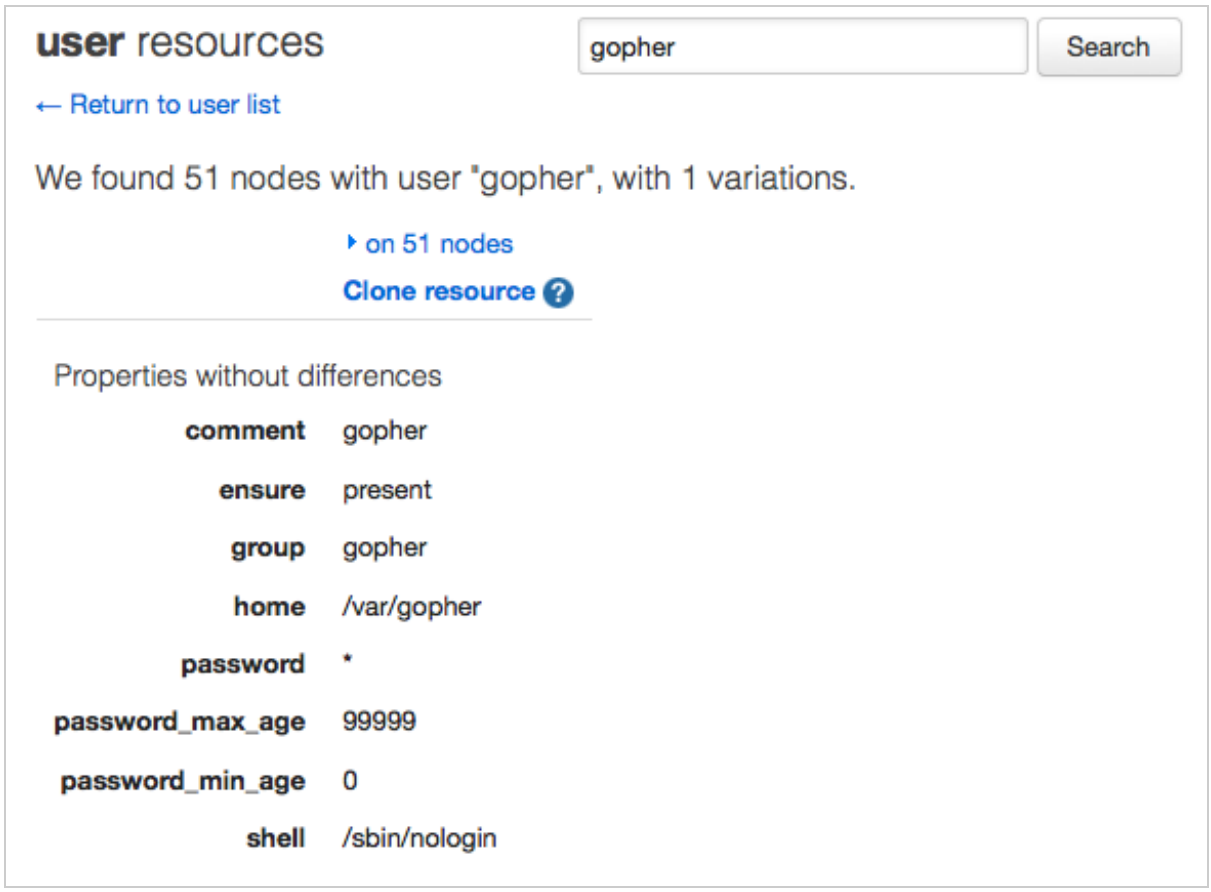


Searching by Name

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.

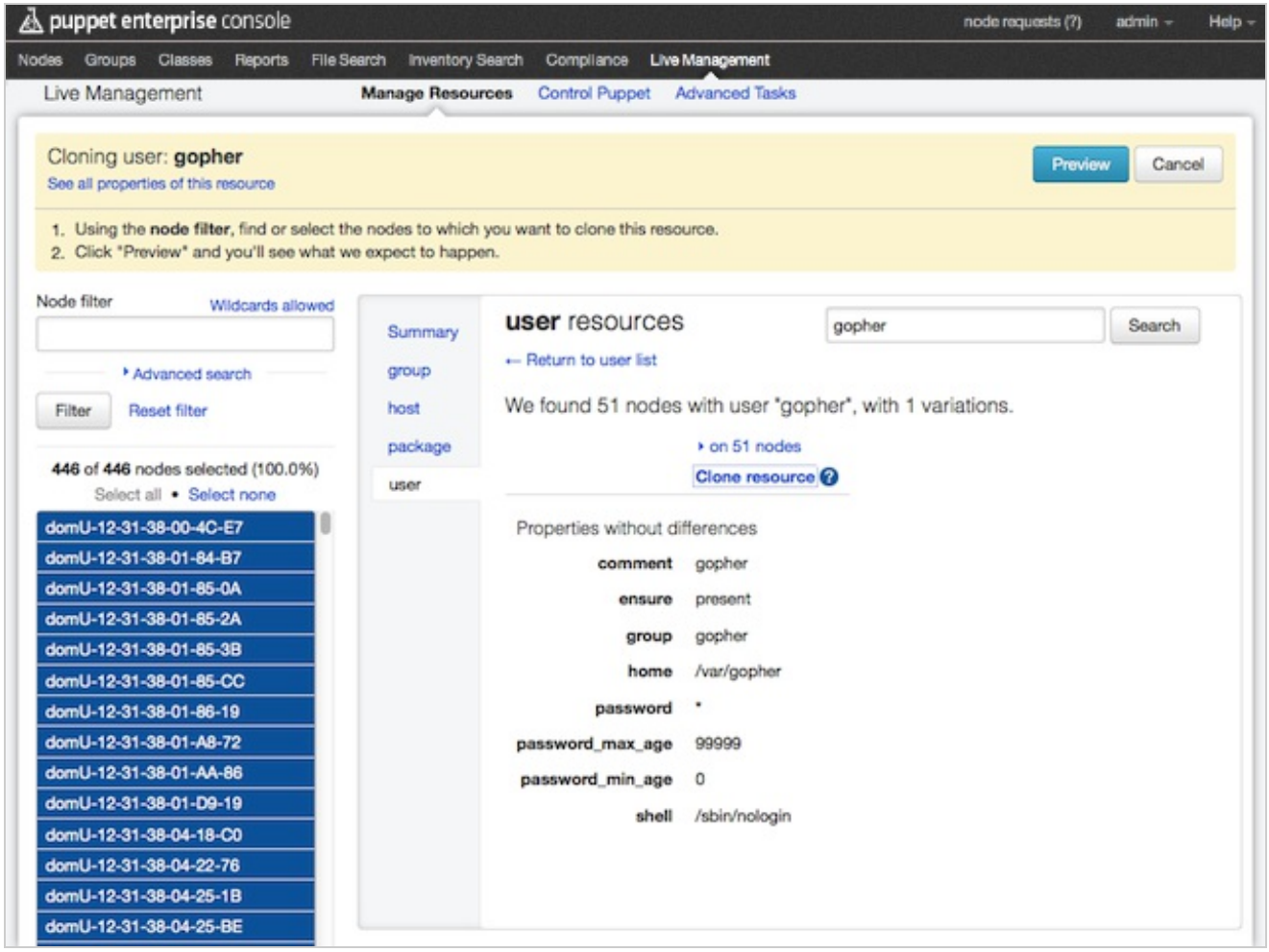
Once you’ve located a resource, the inspect view is the same as that used when browsing.



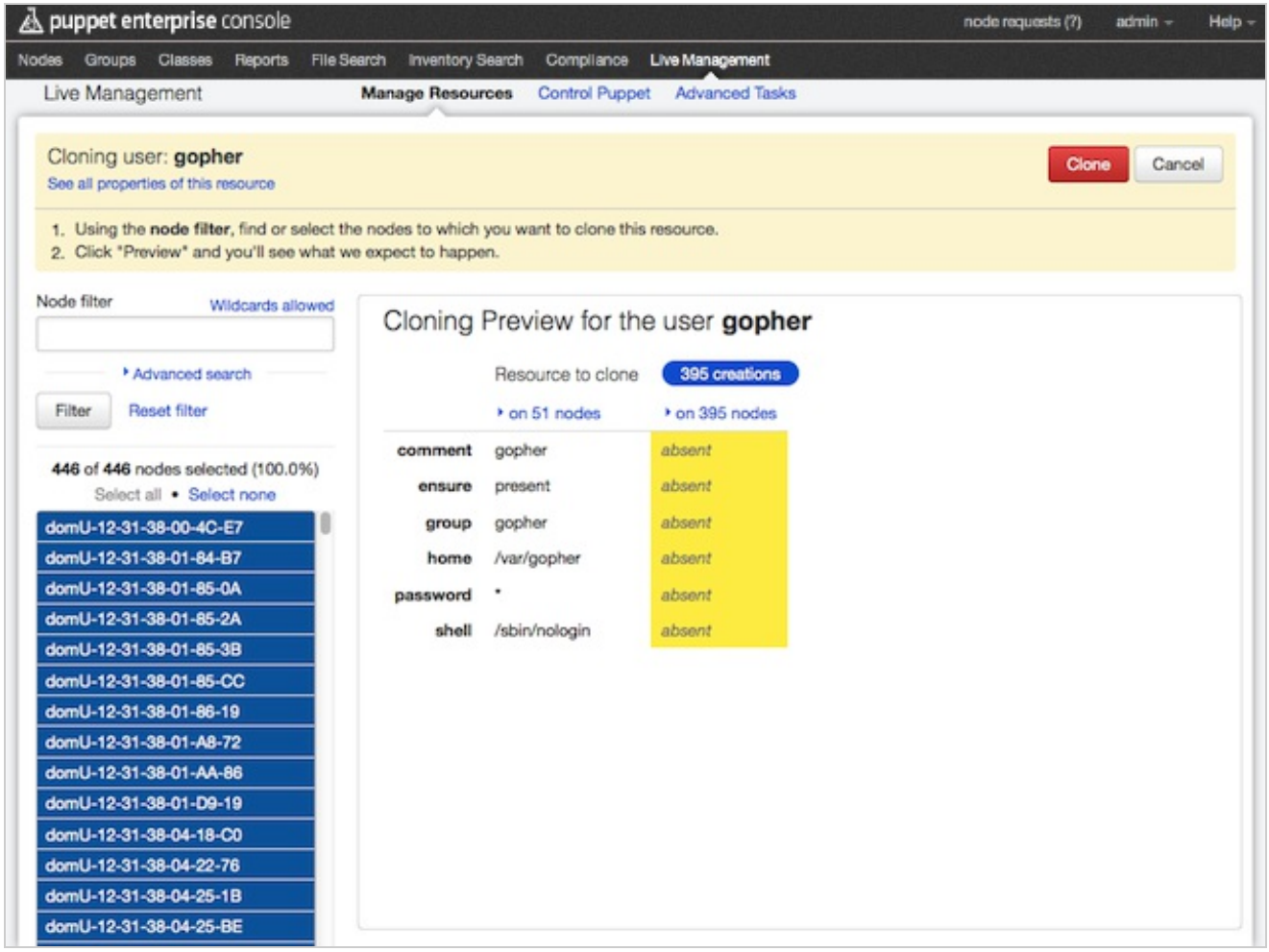
Cloning Resources

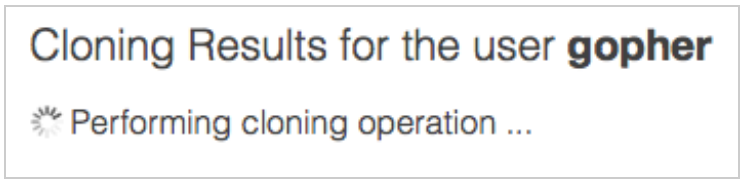
You can use the “clone resource” links on a resource’s inspect view to make it identical on all of the selected nodes. This lets you make your population of nodes more alike without having to write any Puppet code.

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.

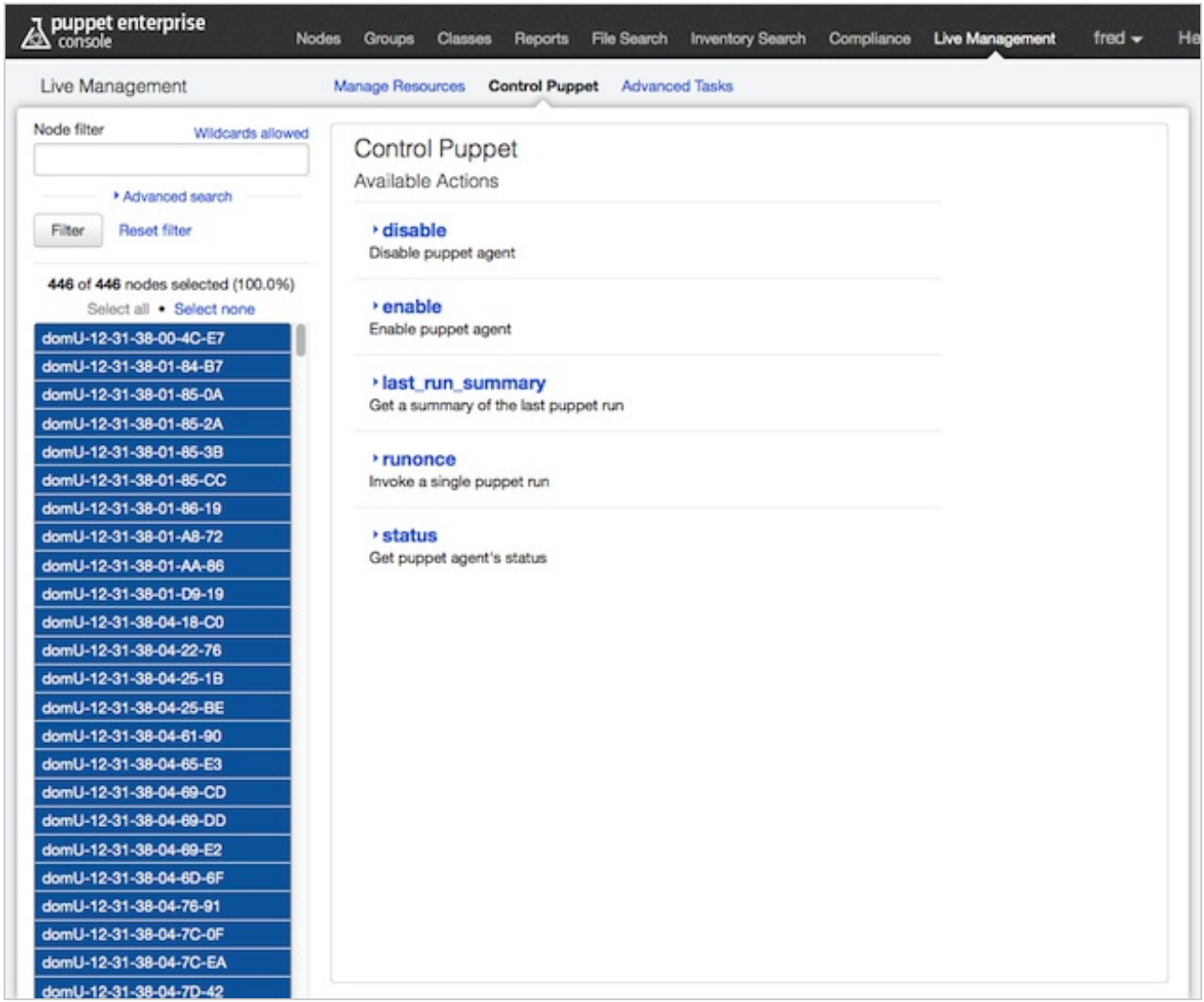
Note also that the UID of a user and the GIDs of its groups aren't cloned across nodes. This means a cloned user's UID will likely differ across nodes. We hope to support UID/GID cloning in a future release.

- Next: Live Management: Controlling Puppet

Live Management: Controlling Puppet

NOTE: Live management and MCollective are not yet supported on Windows nodes.

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.



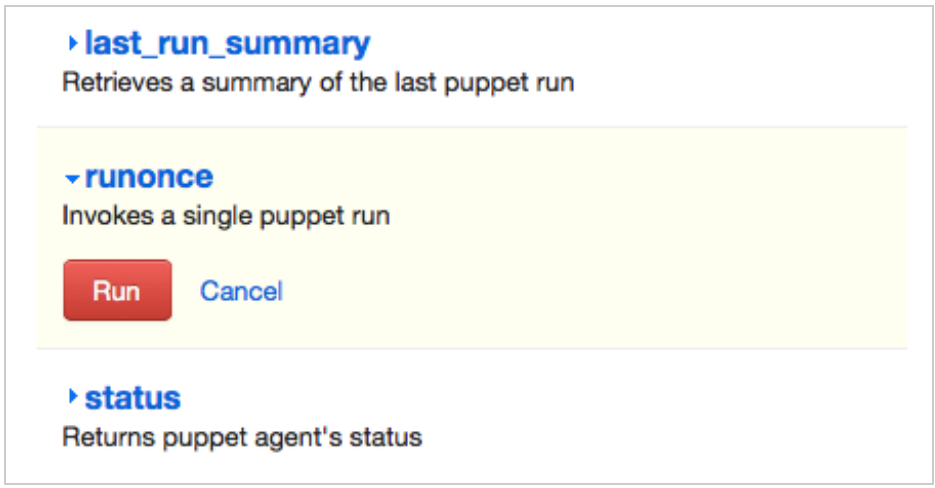
Be aware that the “Control Puppet” tab cannot trigger a node’s first puppet agent run. A node’s first run will happen automatically within 30 minutes after you sign its certificate.

Invoking an Action

The “Control Puppet” tab can perform five actions:

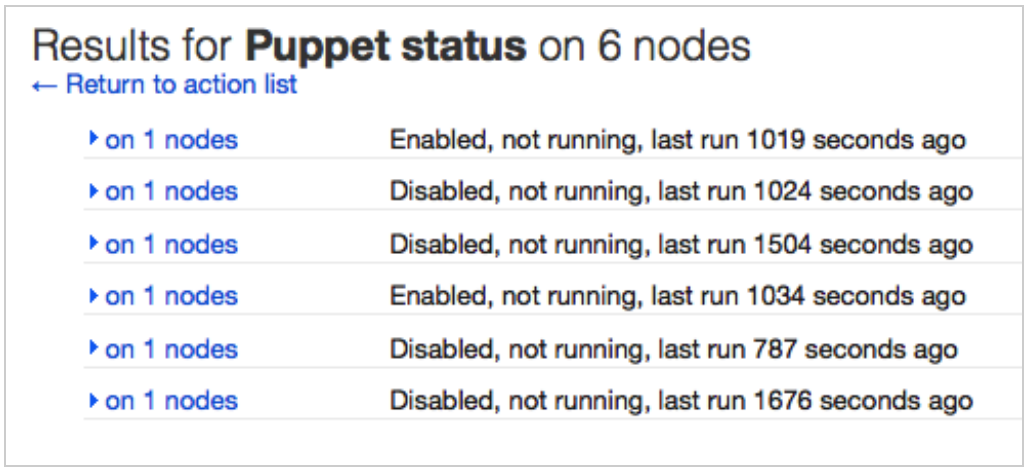
- Disable
- Enable
- Last Run Summary
- Runonce
- Status

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

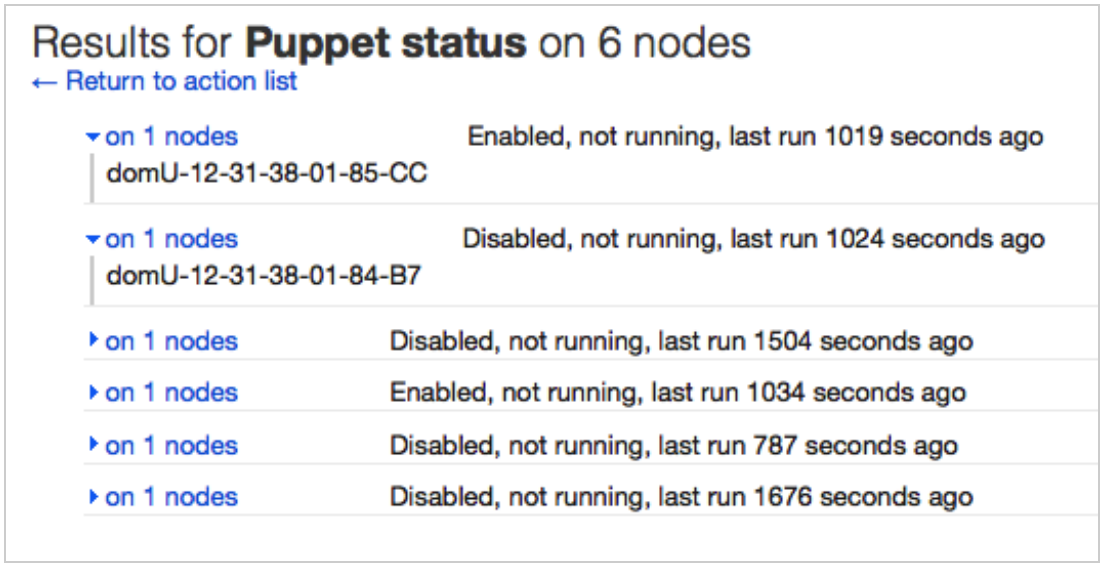


Running an action returns a results view. Results views will attempt to group similar results to reduce noise; click the “on N nodes” links to see which nodes returned a given result.

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

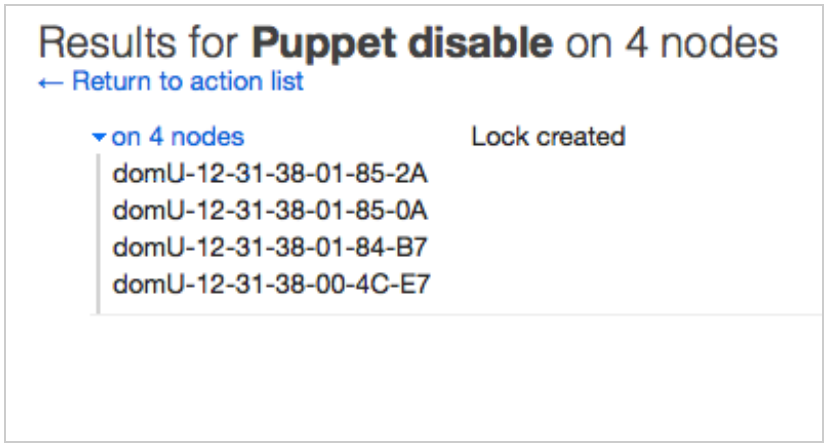
Use the “enable” and “disable” actions to control whether puppet agent does anything. When you disable Puppet on a node, the agent daemon will continue running, but it will not pull configurations from the master.

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.

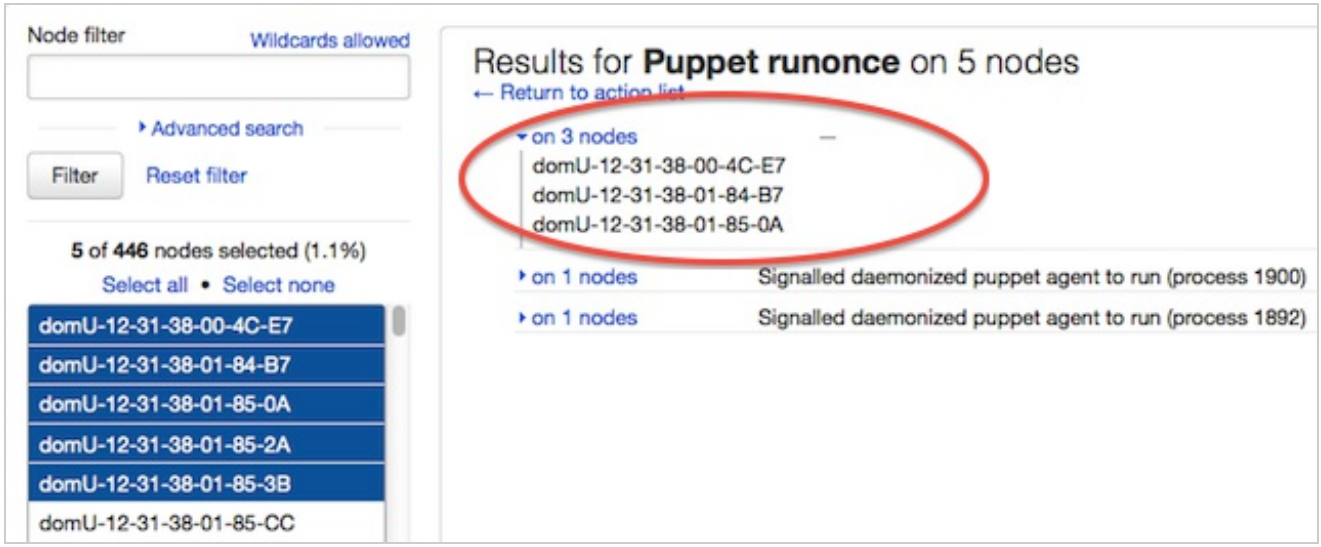
Disabling nodes is great for hedging your bets when you’ve made major changes to an existing Puppet module:

- 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)

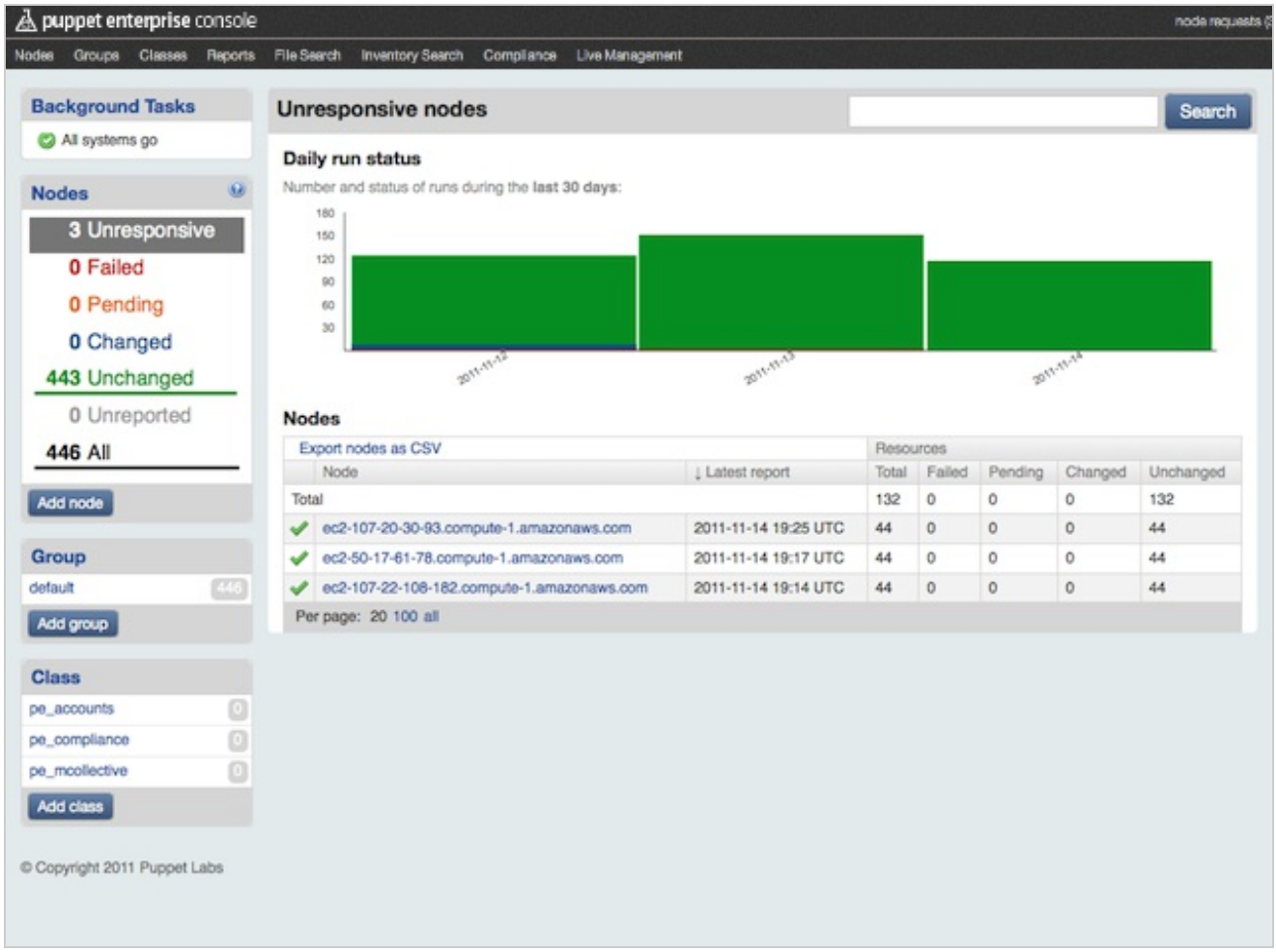
A confirmation screen will appear after disabling some nodes:



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

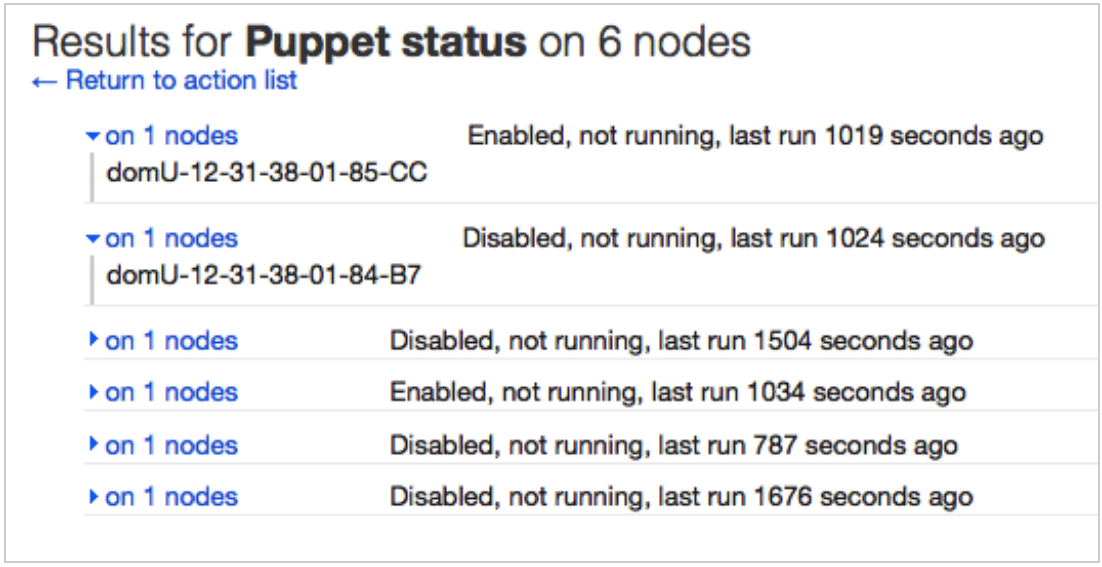
Use the "status" action to check up on puppet agent.

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:

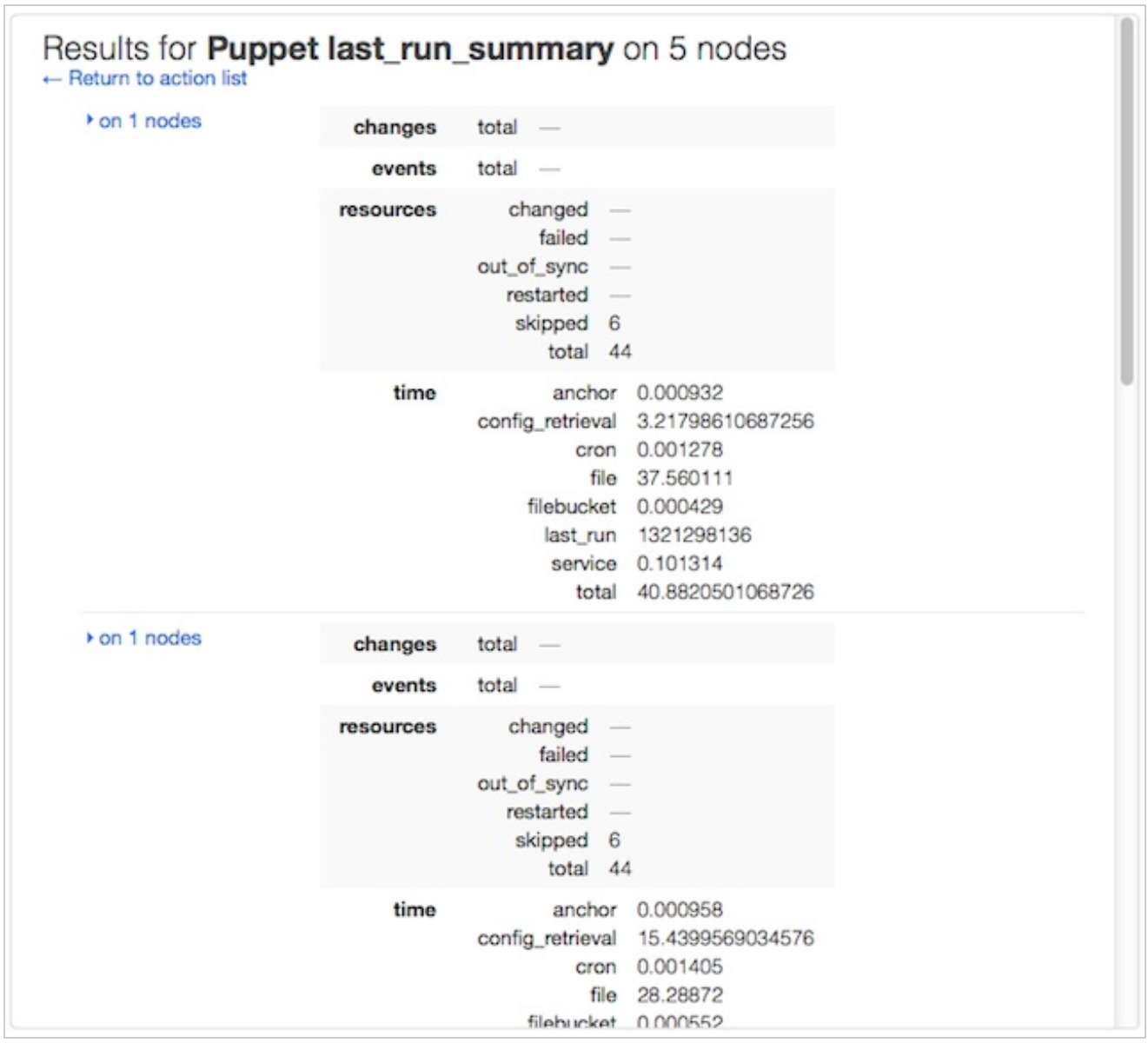


Last Run Summary

Use the “last_run_summary” action for a quick view of what the most recent 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:

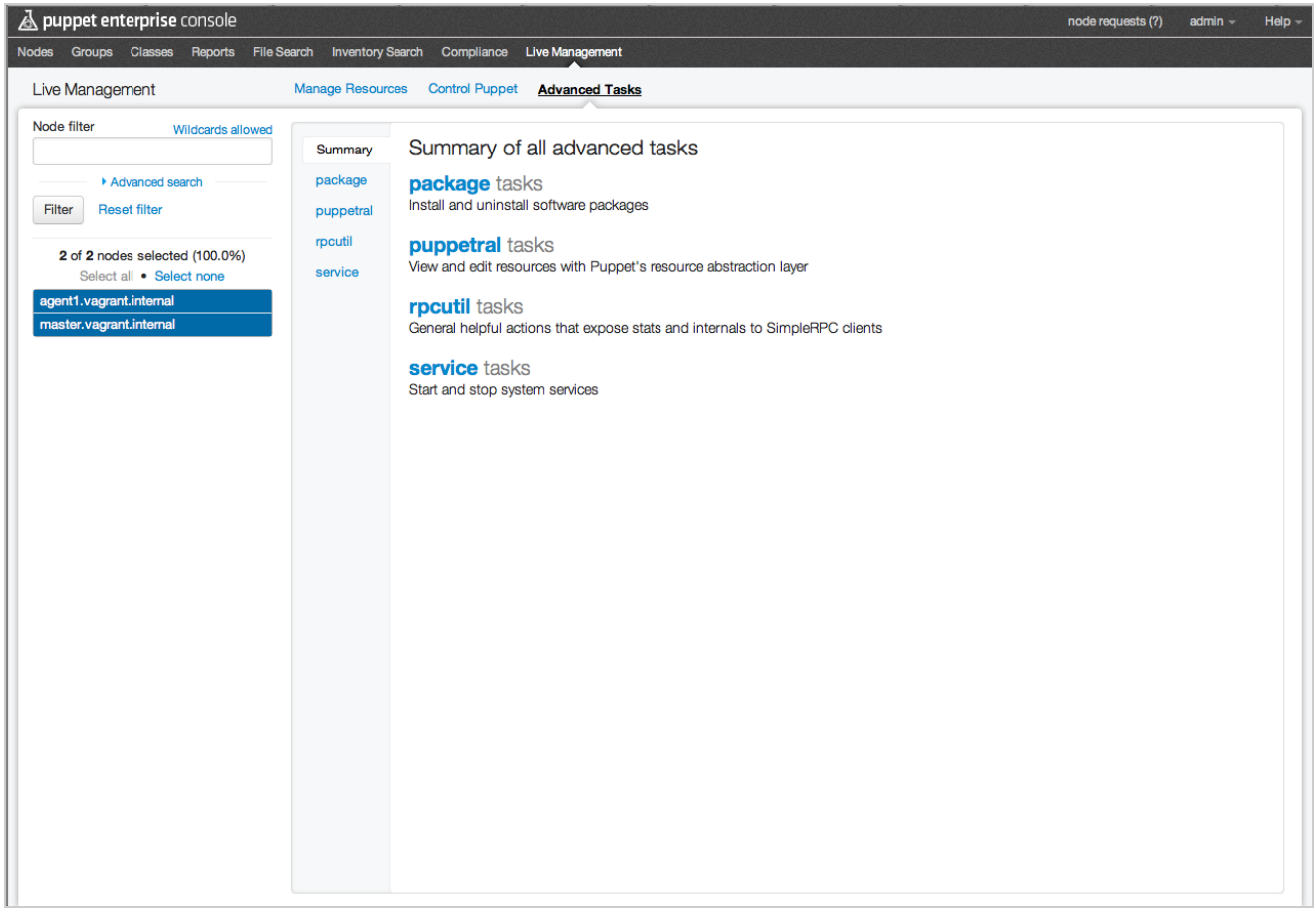


- Next: Live Management: Advanced Tasks

Live Management: Advanced Tasks

NOTE: Live management and MCollective are not yet supported on Windows nodes.

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:

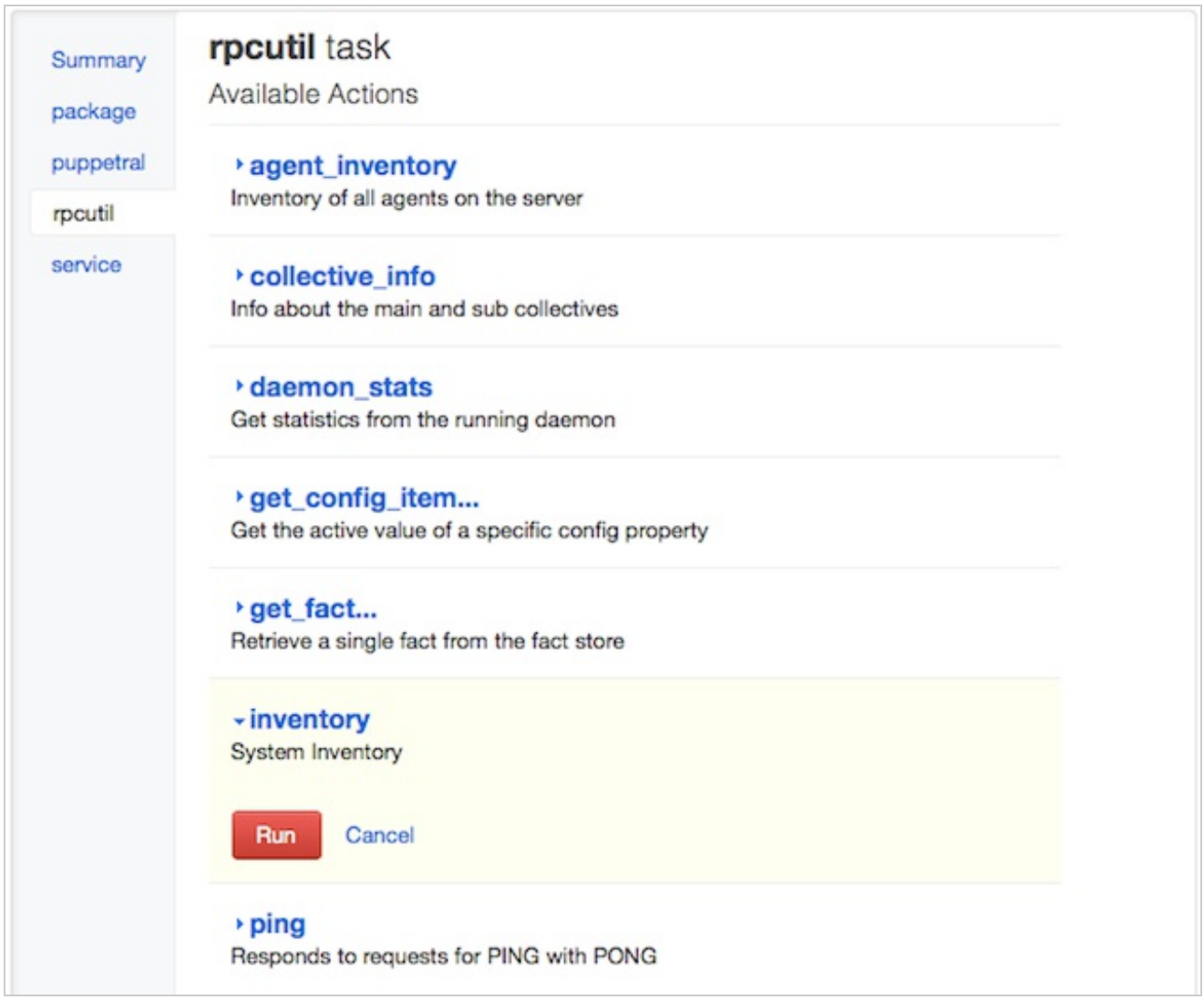
- package — installs and uninstalls software packages.
- rpcutil — performs miscellaneous meta-tasks.
- service — starts and stops services.
- puppetral — the agent used to implement the manage resources tab. When used from the advanced tasks tab, it can only inspect resources, not clone them.

Each agent view includes explanatory text for each action.

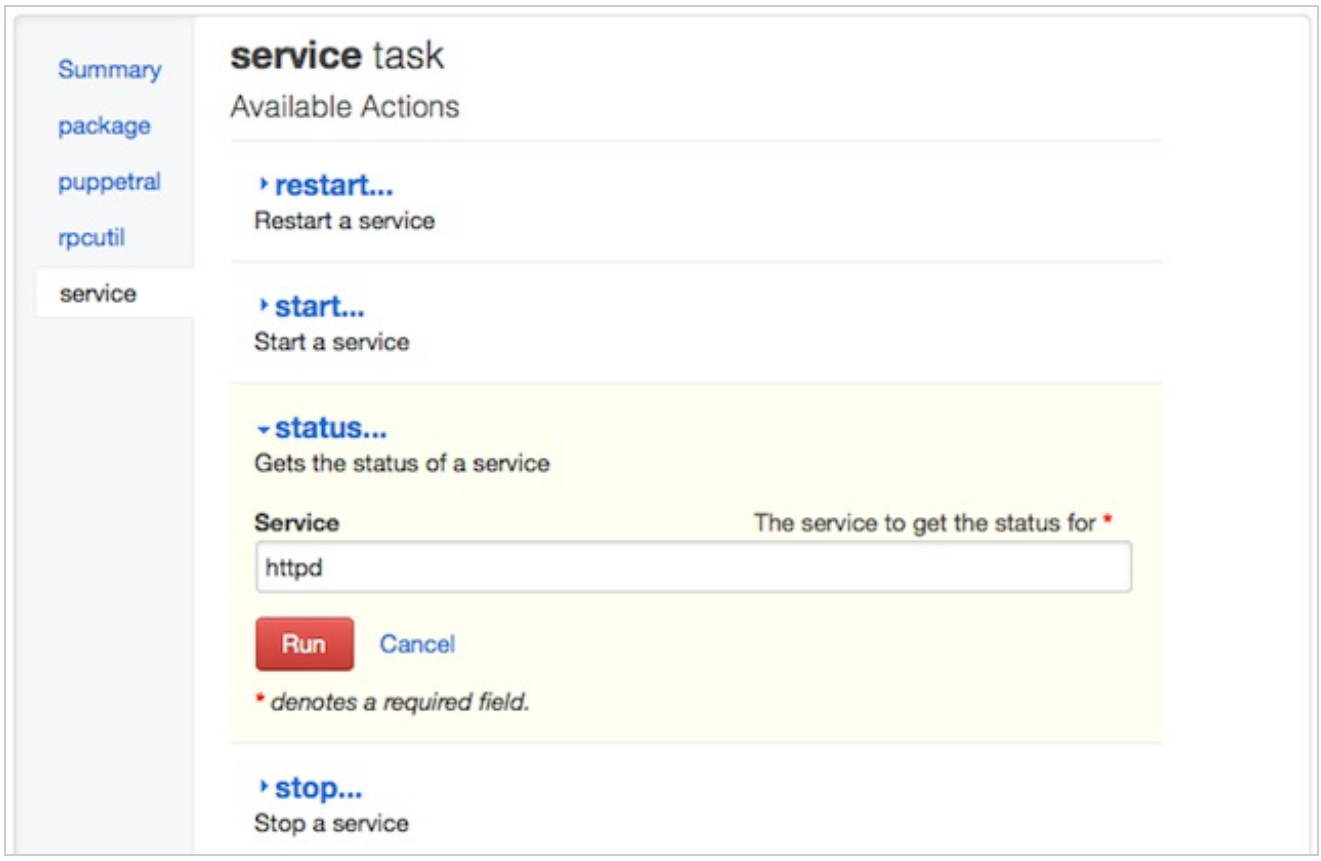
Invoking Actions

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.□

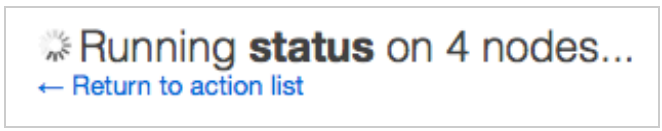
Invoking an action with no arguments:



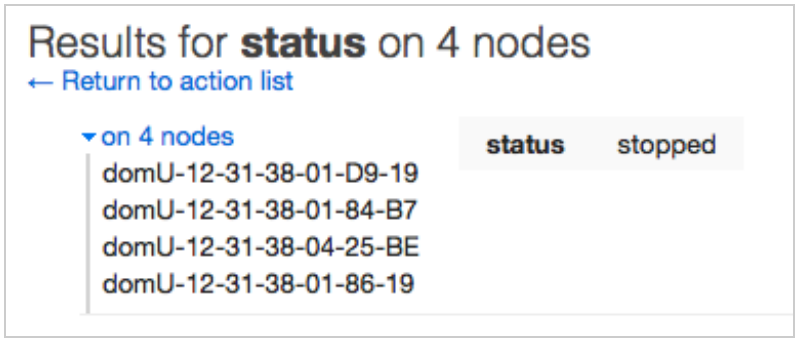
Invoking an action with an argument:



An action in progress:



Results:



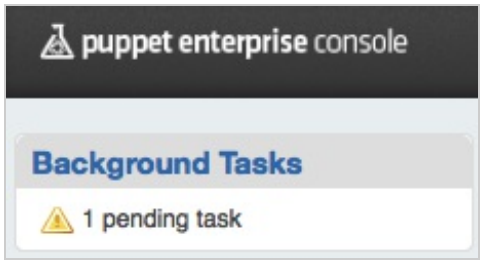
- Next: Console Maintenance

Console Maintenance

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, the background task processes may have 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
```

- Next: Puppet Overview

An Overview of Puppet

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).

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" tab of the console's live management page.


NOTE: Live management and MCollective are not yet supported on Windows nodes.

- Next: Puppet Tools

Puppet Tools

Puppet is built on a large number of services and command-line tools. Understanding which to reach for and when is crucial to using Puppet effectively.

You can read more about any of these tools by running `puppet man <SUBCOMMAND>` at the command line.

Fundamental Services

Puppet agent and puppet master are the heart of Puppet's architecture.

- The puppet agent service runs on every managed Puppet Enterprise node. It fetches and applies configurations from a puppet master server every 30 minutes.

  In Puppet Enterprise, puppet agent runs without user interaction as the `pe-puppet` service. You can manually trigger an immediate run from a node's command line by running `sudo puppet agent --test`. You can also trigger agent runs using the Control Puppet tab of the console's live management page.

  Puppet agent reads its settings from the `[main]` and `[agent]` blocks of `/etc/puppetlabs/puppet/puppet.conf`.

- The puppet master service compiles and serves configurations to agent nodes.

  In Puppet Enterprise, puppet master is managed by Apache and Passenger, under the umbrella of the `pe-httpd` service.

  The puppet master creates agent configurations by consulting its Puppet modules and the instructions it receives from the console.

  The puppet master reads its settings from the `[main]` and `[master]` blocks of `/etc/puppetlabs/puppet/puppet.conf`. It can also be configured conditionally by using environments.

Everyday Tools

- The `puppet cert` subcommand is used to add nodes to your puppet deployment.

  After a new agent node has been installed, it cannot fetch configurations until its certificate request has been approved. Run `puppet cert list` on the puppet master to see which certificate requests are still outstanding, and run `puppet cert sign <NODE>` to approve a request.

  When you decommission a node and remove it from your infrastructure, you should destroy its certificate information by running `puppet cert clean <NODE>` on the puppet master.

- The `puppet apply` subcommand can compile and apply Puppet manifests without the need for a puppet master. It's ideal for testing new modules (`puppet apply -e 'include <CLASS NAME>'`), but can also be used to manage an entire Puppet deployment in a masterless arrangement.
- The `puppet resource` subcommand provides an interactive shell for manipulating Puppet's

underlying resource framework. It is ideal for one-off administration tasks and ad-hoc management, and offers an abstraction layer between various OSs' implementations of core functionality.

```
$ sudo puppet resource package nano ensure=latest
notice: /Package[nano]/ensure: created
package { 'nano':
  ensure => '1.3.12-1.1',
}
```

Advanced Tools

- See the cloud provisioning chapter of this guide for more about the cloud provisioning tools.
- See the orchestration chapter of this guide for more about the command-line orchestration tools.

- Next: Puppet Modules and Manifests

Puppet Modules and Manifests

Puppet configures nodes by reading and applying manifests written by sysadmins. Manifests contain classes, which are chunks of code that configure a specific aspect or feature of a machine.

One or more classes can be stored in a module, which is a self-contained bundle of Puppet code. Pre-existing modules can be downloaded from the Puppet Forge, and most users use a combination of pre-built modules and modules they wrote themselves.

A Fast Introduction to Using Modules

This user's guide includes a pair of interactive quick start guides, which walk you through installing, using, hacking, and creating Puppet modules.

- Quick Start: Using PE
- Quick Start: Writing Modules

A Leisurely Introduction to Writing Puppet Code

For a more complete introduction to Puppet resources, manifests, classes, modules, defined types, facts, variables, and more, read the Learning Puppet series.

- Learning Puppet

Detailed Documentation

The Puppet reference manual contains more information about using modules and the Puppet language.

- Installing and Managing Modules
- Module Fundamentals
- The Puppet Language Guide

Printable References

The module layout cheat sheet is useful when writing your own modules or hacking existing modules.

- Module Layout Cheat Sheet

- Next: Puppet Data Library

The Puppet Data Library

The Puppet Data Library (PDL) consists of two elements:

- The large amount of data Puppet automatically collects about your infrastructure.
- The formats and APIs Puppet uses to expose that data.

Sysadmins can access information from the PDL with their choice of tools, including familiar scripting languages like Ruby, Perl, and Python. Use this data to build custom reports, add to existing data sets, or automate repetitive tasks.

Right now, the Puppet Data Library consists of three different data services:

Puppet Inventory Service

The Puppet Inventory Service provides a detailed inventory of the hardware and software on nodes managed by Puppet.

- Using a simple RESTful API, you can query the inventory service for a node's MAC address, operating system version, DNS configuration, etc. The query results are returned as JSON.
- Inventory information consists of the facts reported by each node when it requests configurations. By installing custom facts on your puppet master server, you can extend the inventory service to contain any kind of data that can possibly be extracted from your nodes.

> EXAMPLE: Using the Puppet Inventory Service, a customer automated the validation and reporting of their servers' warranty status. Their automation used the Puppet Inventory Service to query all servers in the data center on a regular basis and retrieve their serial numbers. These serial numbers are then checked against the server hardware vendor's

warranty database using the vendor's public API to determine the warranty status for each.

Learn more about the Puppet Inventory Service here.

Puppet Run Report Service

The Puppet Run Report Service provides push access to the reports that every node submits after each Puppet run. By writing a custom report processor, you can divert these reports to any custom service, which can use them to determine whether a Puppet run was successful, or dig deeply into the specific changes for each and every resource under management for every node.

You can also write out-of-band report processors that consume the YAML files written to disk by the puppet master's default report handler.

Learn more about the Puppet Run Report Service here.

Puppet Resource Dependency Graph

The Puppet Resource Dependency Graph provides a complete, mathematical graph of the dependencies between resources under management by Puppet. These graphs, which are stored in .dot format, can be used with any commercial or open source visualization tool to uncover hidden linkages and help understand how your resources interconnect to provide working services.

EXAMPLE: Using the Puppet Resource Dependency Graph and Gephi, a visualization tool, a customer identified unknown dependencies within a complicated set of configuration modules. They used this knowledge to re-write parts of the modules to get better performance.

Learn more about the Puppet Resource Dependency Graph here

- Next: Puppet References

Puppet References

Puppet has a lot of moving parts and a lot of information to remember. The following resources will help you keep the info you need at your fingertips and use Puppet effectively.

Terms and Concepts

- The Puppet Glossary defines the common and obscure terms for pieces of the Puppet ecosystem.

Resource Types

Resource types are the atomic unit of Puppet configurations, and there are a lot of them to remember.

- The Core Types Cheat Sheet is a fast, printable two-page guide to the most useful resource types.
- The Type Reference is the complete dictionary of Puppet's built-in resource types. No other page will be more useful to you on a daily basis.

Puppet Syntax

- The Puppet Language Guide covers every part of the Puppet language, from conditionals to parameterized classes.

Configuration and Settings

- Configuring Puppet describes Puppet's configuration files and covers the ten or so most useful settings.
- The Configuration Reference lists every single setting available to Puppet.

- Next: Advanced Configuration

Working with Complex Settings

Many behaviors of Puppet Enterprise can be configured after installation.

Increasing the MySQL Buffer Pool Size

The default MySQL configuration uses an unreasonably small `innodb_buffer_pool_size` setting, which can interfere with the console's ability to function.

If you are installing a new PE 2.7 console server and allowing the installer to configure your databases, it will set the better value for the buffer pool size. However, PE cannot automatically manage this setting for upgraded console servers and manually configured databases.

To change this setting, 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` and possibly as high as `256M`. (Its default value is `8M`, or 8388608 bytes.)

The example diff below illustrates the change to a default MySQL config file:

```
[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
```

After changing the setting, restart the MySQL server:

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

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
    ```

Recovering from a Lost Console Admin Password

If you have forgotten the password of the console's initial admin user, you can create a new admin user and use it to reset the original admin user's password.

On the console server, run the following commands:

```
$ cd /opt/puppet/share/console-auth
$ sudo /opt/puppet/bin/rake db:create_user EMAIL="adminuser@example.com"
PASSWORD="<password>" ROLE="Admin"
```

You can now log in to the console as the user you just created, and use the normal admin tools to reset other users' passwords.

Note: in PE 2.5.1 and 2.5.0, use the following command instead:

```
$ sudo /opt/puppet/bin/rake -f /opt/puppet/share/console-auth/Rakefile
db:create_initial_admin email=user@example.com password=<PASSWORD>
```

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
   ```

Configuring Console Authentication

Configuring the SMTP Server

The console's account system sends verification emails to new users, and requires an SMTP server to do so. If your site's SMTP server requires a user and password, TLS, or a non-default port, you can configure these by editing the `/etc/puppetlabs/console-auth/config.yml` file:

```
smtp:
  address: mail.example.com
  port: 25
  use_tls: false
  ## Uncomment to enable SMTP authentication
  #username:  smtp_username
  #password:  smtp_password
```

Allowing Anonymous Console Access

To allow anonymous, read-only access to the console, do the following:

- Edit the `/etc/puppetlabs/console-auth/cas_client_config.yml` file and change the `global_unauthenticated_access` setting to `true`.
- Restart Apache by running `sudo /etc/init.d/pe-httpd restart`.

Disabling and Reenabling Console Auth

If necessary, you can completely disable the console's access controls. Run the following command to disable console auth:

```
$ sudo /opt/puppet/bin/rake -f /opt/puppet/share/console-auth/Rakefile
console:auth:disable
```

To re-enable console auth, run the following:

```
$ sudo /opt/puppet/bin/rake -f /opt/puppet/share/console-auth/Rakefile
console:auth:enable
```

Using LDAP or Active Directory Instead of Console Auth

For instructions on using third-party authentication services, see the console_auth configuration page.

Changing Orchestration Security

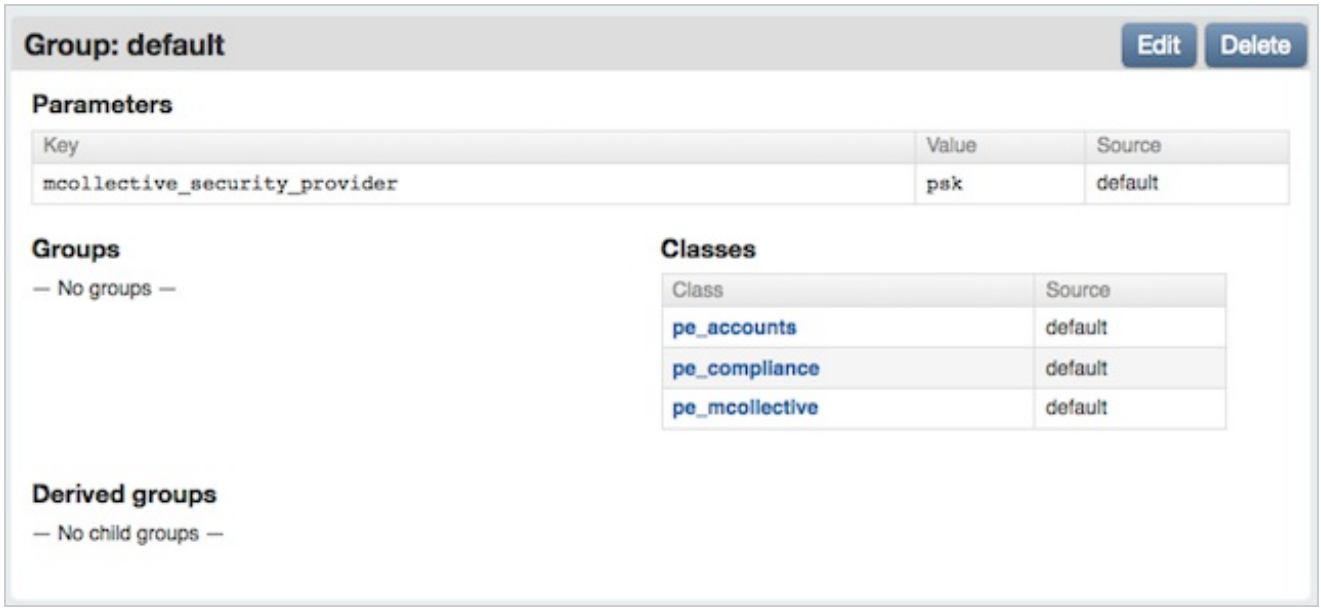
PE's orchestration messages are encrypted and authenticated. You can easily change the authentication method or the password used in the PSK authentication method.

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.□

Fine-tuning the `delayed_job` Queue

The console uses a `delayed_job` queue to asynchronously process resource-intensive tasks such as report generation. Although the console won't lose any data sent by puppet masters if these jobs don't run, you'll need to be running at least one delayed job worker (and preferably one per CPU core) to get the full benefit of the console's UI.□

Currently, to manage the `delayed_job` workers, you must either use the provided monitor script or start non-daemonized workers individually with the provided rake task.

Using the monitor script

The console ships with a worker process manager, which can be found at `script/delayed_job`. This tool's interface resembles an init script, but it can launch any number of worker processes as well as a monitor process to babysit these workers; run it with `--help` for more details.

`delayed_job` requires that you specify `RAILS_ENV` as an environment variable. To start four worker processes and the monitor process:

```
# env RAILS_ENV=production script/delayed_job -p dashboard -n 4 -m start
```

In most configurations, you should run exactly as many workers as the machine has CPU cores.

Tuning the ActiveMQ Heap Size

The puppet master node runs an ActiveMQ server to support orchestration commands. By default, the ActiveMQ process uses a Java heap size of 512 MB, which has been tested to support thousands of nodes.

You can increase or reduce the amount of memory used by ActiveMQ by navigating to the puppet master node's page in the console and creating a new parameter called `activemq_heap_mb`. The value you assign to it will be the amount of memory, in megabytes, used by ActiveMQ; delete the parameter to revert to the default setting.

This is most commonly used to create stable proof-of-concept deployments on virtual machines with limited amounts of RAM. Many of the puppet master's features can fail if ActiveMQ consumes all of the available memory on the system, and reducing its heap size by half or more can prevent these problems on a starved VM.

Setting ActiveMQ Thread Pooling

By default, ActiveMQ is set up to use a dedicated thread for every destination. In environments with large numbers of destinations, this can cause memory resource issues. If the ActiveMQ log is full of "java.lang.OutOfMemoryError: unable to create new native thread" errors, you can configure ActiveMQ to use a thread pool by setting the system property: `-Dorg.apache.activemq.UseDedicatedTaskRunner=false`. This is specified in the ActiveMQ start script via ACTIVEMQ_OPTS. Using a thread pool will reduce the number of threads required by ActiveMQ and so should reduce its memory consumption.

- Next: Installing Additional Components

Installing Additional Components

Several optional components of PE can be installed manually.

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*
```

- Next: Orchestration Overview

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 installed on agent nodes that 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.)

 NOTE: Orchestration and MCollective are not yet supported on Windows nodes.

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, using 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 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.

- Next: Orchestration Usage and Examples

Orchestration for New PE Users: Usage and Examples

 NOTE: Orchestration and MCollective are not yet supported on Windows nodes.

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 will 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 2.7 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 forces the find action to only return nodes which 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 than 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 than 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 than 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
```

The creation of this file 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 shown 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` plugin manages software packages outside of a normal Puppet configuration run. The following is the trimmed output from 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
```

- Next: Cloud Provisioning Overview

A High Level Look at Puppet's Cloud Provisioning Tools

Puppet Enterprise includes a suite of command-line tools you can use for provisioning new virtual nodes when building or maintaining cloud computing infrastructures based on VMware VSphere and Amazon EC2. You can use these tools to:

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

When used together, these tools provide quick and efficient workflows for adding and maintaining fully configured, ready-to-run virtual nodes in your Puppet Enterprise-managed cloud environment.

See the sections on VMware, and AWS provisioning for details about creating and destroying virtual machines in these environments. Beyond that, the section on classifying nodes and installing PE covers actions that work on any new machine, virtual or physical, in a cloud environment. To get an idea of a typical workflow in a cloud provisioning environment, see the Workflow section.

The cloud provisioning tools can be added during installation of Puppet Enterprise. If you have already installed PE, and you want to install the cloud provisioning tools, simply run the upgrader again.

Tools

PE's provisioning tools are built on the `node`, `node_vmware`, and `node_aws` subcommands. Each of these subcommands has a selection of available actions (such as `list` and `start`) that are used to complete specific provisioning tasks. You can get detailed information about a subcommand and its actions by running `puppet help` and `puppet man`.

The VMware and AWS subcommands are only used for cloud provisioning tasks. `Node`, on the other hand, is a general purpose Puppet subcommand that includes several provisioning-specific actions. These are:

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

The `clean` action may also be useful when decommissioning nodes.

All of the cloud provisioning tools are powered by Fog, the Ruby cloud services library. Fog is automatically installed on any machine receiving the cloud provisioner role.

- Next: Installing and Configuring Cloud Provisioner

Installing and Configuring Cloud Provisioning

There are many options and actions associated with the main cloud provisioning sub-commands, `node`, `node_vmware`, `node_aws` and `node_openstack`. This page provides an overview but, for all the details, check the man pages.

Prerequisites

Software

- The cloud provisioning tools ship with Puppet Enterprise 2.0 and later. OpenStack tools require cloud_provisioner 1.0.2 or later (i.e. PE 2.5 or later).
- Puppet 2.7.2 or later is required
- The Fog cloud services library must be installed on the machine running cloud_provisioner. (Some operating systems and environments may also require you to manually install some of Fog's dependencies.) To install Fog, run

  ```
  # gem install fog -v 0.7.2
  ```

- Cloud_provisioner also requires the GUID library so it can generate unique indentifiers.

```
# gem install guid
```

Note that later versions of fog may not be fully compatible with Cloud Provisioner. This issue is currently being addressed.

Services

The following services and credentials are required:

- VMware requires: VMware vSphere 4.0 (or later) and VMware vCenter
- Amazon Web Services requires: An existing Amazon account with support for EC2
- OpenStack requires: A standard installation of OpenStack Keystone and the accompanying EC2 credentials

Installing

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

The Puppet Enterprise installer will ask whether or not to install cloud provisioning during installation. Answer 'yes' to enable cloud provisioning actions on a given node.

If you have already installed PE without installing the cloud provisioning tools, run the upgrader script `puppet-enterprise-upgrader` and answer "yes" when prompted to install the tools. Running the upgrader script will have no ill effects on your current installation, even if the upgrader is for the version currently installed. (No user-configured files will get overwritten, and of course the installer backs up all relevant files as well.)

If you're using an answer file to install Puppet Enterprise, you can install cloud provisioning by setting the `q_puppet_cloud_install` option to `y`.

Configuring

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

Start by creating a file called `.fog` in the home directory of the user who will be provisioning new nodes.

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

This will be the configuration file for Fog, the cloud abstraction library that powers PE's provisioning tools. Once it is filled out, it will consist of a YAML hash indicating 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
```

See below to learn 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 entering the other three settings and then 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
```

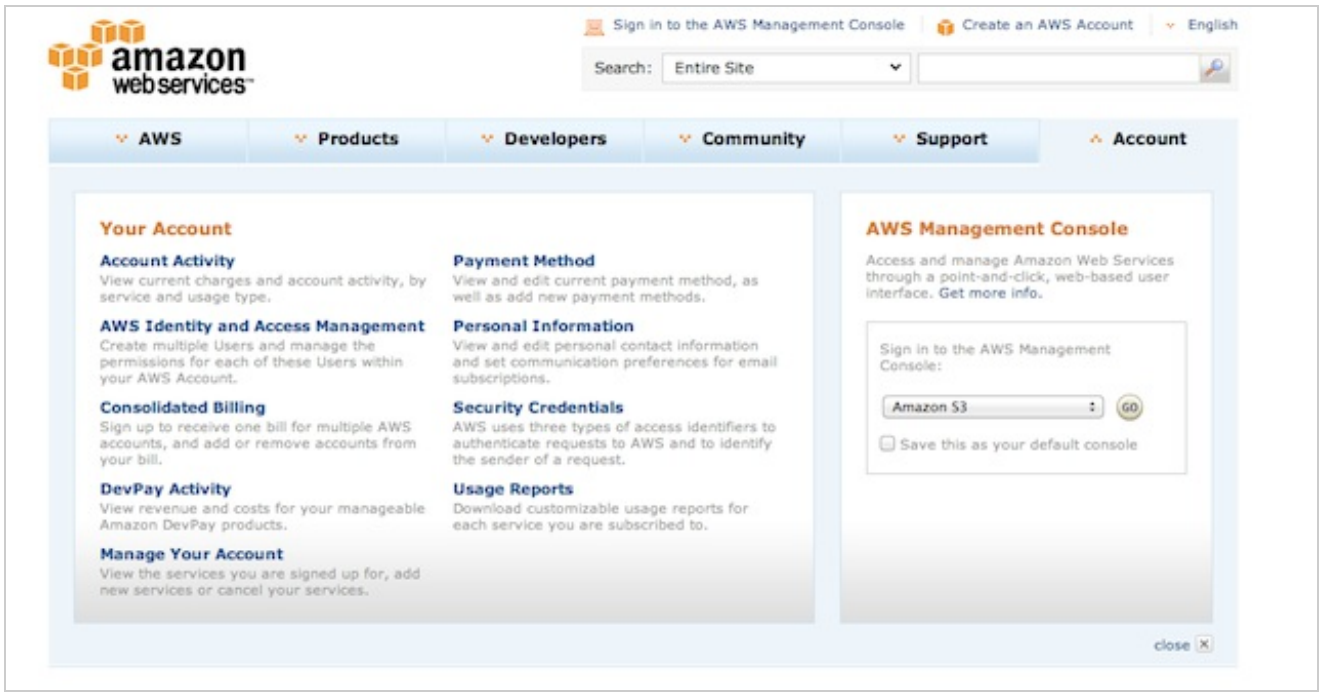
…which can then be entered as the value of this setting.

Adding Amazon Web Services or OpenStack Credentials

To connect to Amazon Web Services or your OpenStack server, 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.

For AWS installations, you can 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 choose “Access Credentials.” 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, click the “Show” link under “Secret Access Key”.

For OpenStack installations, your credentials are printed to screen after running `keystone ec2-credentials-create`

Put both keys in your `~/.fog` file as described above. You will also need to generate an SSH private key using Horizon, or simply import a selected public key.

Additional AWS Configuration

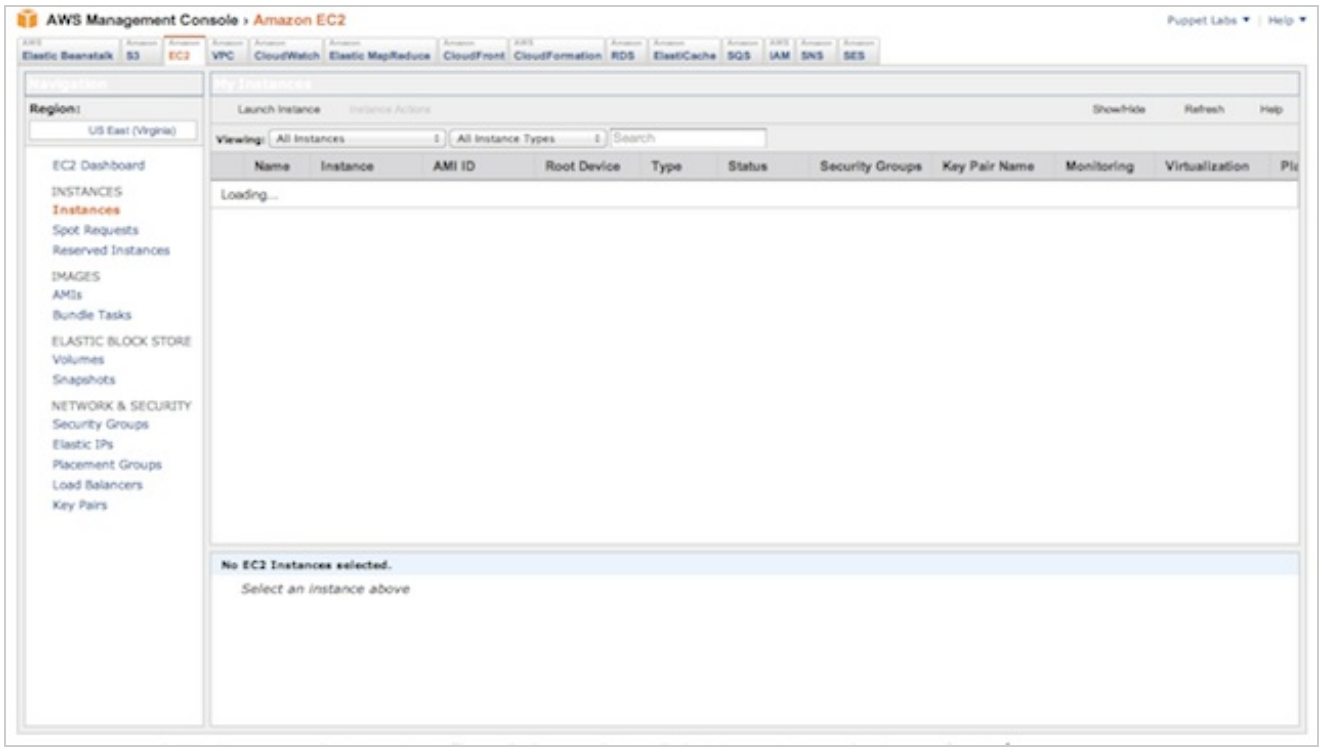
To provision with Puppet, your Amazon Web Services EC2 account will need to have:

- 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.

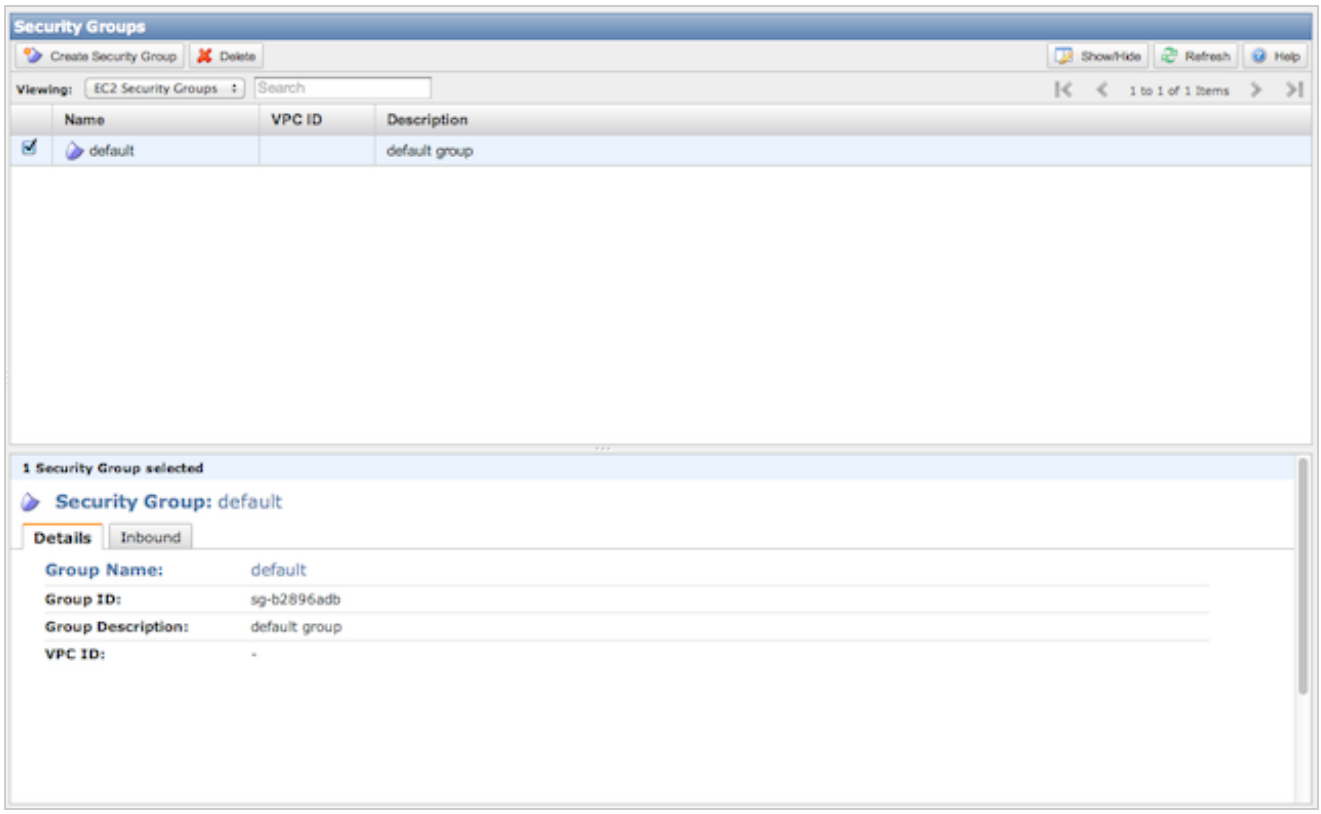


Select the “Key Pairs” menu item from the dashboard. If you don’t have any existing key pairs, you can create one with the “Create Key Pairs” button. Specify a new name for the key pair to create it; the private key file will be automatically downloaded to your host.

Make a note of the name of your key pair, since you will need to know it 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. Make sure that inbound SSH traffic is using port 22. You can also indicate 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.

Additional OpenStack Configuration

Before you can launch any instances with the provisioner module, you will need:

- a fully functional OpenStack environment
- at least one valid OS image
- the URL of your Nova EC2 API server (typically, http://your.nova.api.server:8773/services/Cloud)
- to use the Horizon console to configure the default security group to allow SSH (port 22) access.

Demonstration

The following video demonstrates the setup process and some basic functions:

- Next: Classifying Nodes and Remotely Installing Puppet

Classifying New Nodes and Remotely Installing Puppet

Nodes in a cloud infrastructure can be classified and managed as easily as any other machine in a puppet environment. You can add new nodes to pre-existing console groups, further classify and configure those nodes, and install Puppet (Enterprise or open source) on them.

Many of these tasks are accomplished using the `puppet node` subcommand. While `puppet node` can be applied to physical or virtual machines, several actions have been created specifically for working with virtual machine instances in the cloud. For complete details, view the `puppet node` man page and the Getting Started With Cloud Provisioner page.

Classifying nodes

Once you have created instances for your cloud infrastructure, you need to start configuring them and adding the files, settings and/or services needed for their intended purposes. The fastest and easiest way to do this is to add them to your existing console groups. You can do this by adding nodes to a group with the console's web interface. However, you can also do this right from the command line, which can be more convenient if you're already at your terminal and have the node's name ready at hand.

To classify nodes and add them to a console group, run `puppet node classify`.

```
$ 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
```

The above example adds an AWS EC2 instance to the console. Note that you use the name of the node you are classifying as the command's argument and the `--node-group` option to specify the group you want to add your new node to. The other options contain the connection and authentication data needed to properly connect to the node.

Note that until the first puppet run is performed on this node, puppet itself will not yet be installed. (Unless one of the "wrapper" commands has been used. See below.)

To see additional help for node classification, run `puppet help node classify`. For more about how the console groups and classifies nodes, see the section on grouping and classifying.

You may also wish review the basics of Puppet classes and configuration to help you understand how groups and classes interact.

The process of adding a node to the console is demonstrated in the following video:

Installing Puppet

Use the `puppet node install` command to install Puppet Enterprise (or open-source puppet) onto the new instances.

```
$ 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
```

This command's options specify:

- The path to a private SSH key that can be used log in to the vm, specified with the `--keyfile` option. The `install` action uses SSH to connect to the host, and so needs access to an SSH key. For Amazon EC2, point to the private key from the key pair you used to create the instance. In most cases, the private key is in the ~/.ssh directory. (Note that for VMware, the public key should have been loaded onto the template you used to create your virtual machine.)
- The local user account used to log in, specified with the `--login` option.

For the command's argument, specify the name of the node on which you're installing Puppet Enterprise.

For the default installation, the `install` action uses the installation packages provided by Puppet Labs and stored in Amazon S3 storage. You can also specify packages located on a local host, or on a share in your local network. Use `puppet help node install` or `puppet man node` to see more details.

In addition to these default configuration options, you can specify a number of additional options to control how and what we install on the host. You can specify the specific version of Puppet Enterprise to install (or you can install open source puppet). You 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.

The process of installing Puppet on a node is demonstrated in detail in the following video:

Classifying and Installing Puppet in One Command

Using `node init`

Rather than using multiple commands to classify and install Puppet on a node, there are a couple of other options that combine actions into a “wrapper” command. Note that you will need access to the PE installer, which is typically specified with the `--installer-payload` argument.

If a node has been prepared to remotely sign certificates, you can use the `init` action which will `install` Puppet, `classify` the node and sign the certificate in one step. 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 invocation above will connect to the console, classify the node in the `appserver_pool` group, and then install Puppet Enterprise on this node.

Using `autosign.conf`

Alternatively, if your deployment has been set up to use the `autosign` configuration parameter, you can use it to sign the certifcate automatically. While this can greatly simplify the process, there are some possible security risks associated with going this route, so be sure you are comfortable with

the process and know the risks.

- Next: Provisioning with VMware

Provisioning With VMware

Puppet Enterprise provides support for working with VMware virtual machine instances using vSphere and vCenter. Using actions of the `puppet node_vmware` sub-command, you can create new machines, view information about existing machines, classify and configure machines, and tear machines down when they're no longer needed.

The main actions used for vSphere cloud provisioning include:

- `puppet node_vmware list` for viewing existing instances
- `puppet node_vmware create` for creating new instances
- `puppet node_vmware terminate` for destroying no longer needed instances.

If you're new to VMware vSphere, you should start by looking at the vSphere documentation.

Listing VMware vSphere Instances

Let's get started by listing the machines currently on our vSphere server. You 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 the correct, trusted vSphere server by checking the hostname in your `~/.fog` file, then add the hash to the `.fog` file as follows:

```
:vsphere_expected_pubkey_hash:
431dd5d0412aab11b14178290d9fcc5acb041d37f90f36f888de0cebfffff0a8
```

Now you should be able to run the `puppet node_vmware list` command and see a list of 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
```

This shows that you're connected to your vSphere server and shows an available VMware template (at `master_template`). Also shown are two virtual machines (puppetmaster.example.com and agent.example.com). VMware templates contain the information needed to build new virtual machines, such as the operating system, hardware configuration, and other details.

Specifically, `list` will return all of the following information:

- 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 using 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%
|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo| Time: 00:00:03
---
name: newpuppetmaster
power_state: poweredOff
...
status: success
```

Here `node_vmware create` has built a new virtual machine named `newpuppetmaster` with a template of `/Datacenters/Solutions/vm/master_template`. (This is the template seen earlier with the `list` action.) The virtual machine will be powered on, which may take several minutes to complete.

The following video demonstrates the above and some other basic functions:

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, use:

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

This will stop the running virtual machine (which 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 and permanent action that should only be taken when you wish to delete the virtual machine and its data!

The following video demonstrates the termination process and some other related functions:

Getting more help

The `puppet node_vmware` command has extensive in-line help 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 get help on individual actions by running:

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

For example:

```
$ puppet help node_vmware start
```

- Next: Provisioning with AWS

Provisioning With Amazon Web Services

Puppet Enterprise provides support for working with EC2 virtual machine instances using Amazon Web Services. Using actions of the `puppet node_aws` sub-command, you can create new machines, view information about existing machines, classify and configure machines, and tear machines down when they're no longer needed.

The main actions used for AWS cloud provisioning include:

- `puppet node_aws list` for viewing existing instances

- `puppet node_aws create` for creating new instances
- `puppet node_aws terminate` for destroying no longer needed instances

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

Below we take a quick look at these actions and their associated options. For comprehensive information, see Getting More Help below.

Viewing Existing EC2 Instances

Let's start by finding out about the currently running EC2 instances. You 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
```

This shows three running EC2 instances. For each instance, the following characteristics are shown:

- The instance name
- The date the instance was created
- 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, nothing will be returned.

Creating a new EC2 instance

New instances are created using the `node_aws create` or the `node_aws bootstrap` actions. The `create` action simply builds a new EC2 machine instance. The `bootstrap` "wrapper" action creates, classifies, and then initializes the node all in one command.

Using `create`

The `node_aws create` subcommand is used to build a new EC2 instance based on a selected AMI image.

The subcommand has three required options:

- The AMI image we'd like to use. (`--image`)
- The name of the SSH key pair to start the image with (`--keyname`). See here for more about creating Amazon-managed key pairs.
- The type of machine instance we wish to create (`--type`). You can see a list of types here.

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
```

You've created a new instance using an AMI of `ami-edae6384`, a key named `cloudprovisioner`, and of the machine 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 in which to start the instance. 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`.

Using `bootstrap`

The `bootstrap` action is a wrapper that combines several actions, allowing you to create, classify, install Puppet on, and sign the certificate of EC2 machine instances. Classification is done via the Console.

In addition to the three options required by `create` (see above), `Bootstrap` also requires the following:

- The name of the user Puppet should be using when logging in to the new node. (`--login` or `--username`)
- The path to a local private key that allows SSH access to the node (`--keyfile`). Typically, this is the path to the private key that gets downloaded from the Amazon EC2 site.

The example below will bootstrap a node using the ami-0530e66c image, located in the US East region and running as a t1.micro machine type.

```
puppet node_aws bootstrap
--region us-east-1
--image ami-0530e66c
--login root --keyfile ~/.ec2/ccaum_rsa.pem
--keyname ccaum_rsa
--type t1.micro
```

Demo

The following video demonstrates the EC2 instance creation process in more detail:

Connecting to an EC2 instance

You connect to EC2 instances via SSH. To do this you will need the private key 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
```

You 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, permanent action that should only be performed when you are confident the instance, and its data, are no longer needed.

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
```

The following video demonstrates the EC2 instance termination process in more detail:
</param></param></param></embed>

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.
```

For more detailed help you can also view the man page .

```
$ 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
```

- Next: Sample Cloud Provisioning Workflow

A Day in the Life of a Puppet-Powered Cloud Sysadmin

Tom is a sysadmin for CloudWidget.com, a company that provides web-based application services. They use a three-tier application architecture with the following types of nodes:

1. A web front-end load balancer
2. A pool of application servers behind the load balancer
3. A database server that serves the application servers

All of these nodes are virtual machines running on a VMware ESX server. The nodes are all currently being managed with Puppet Enterprise. Using PE, the application servers have all been assigned to a group which applies a class `cloudwidget_appserv`.

CloudWidget is growing rapidly and so Tom is not surprised when he checks his inbox and finds several messages from users complaining about sluggish performance. He checks his monitoring

tool and sure enough the load is too high on his application servers and performance is suffering. It's time to add a new node to the application server pool to help better distribute the load.

Tom grabs a cup of coffee and fires up his terminal. He starts by creating a new virtualized node with `puppet node_vmware create`. This gives him a new node with the following characteristics:

- The node has a complete OS already installed
- The node includes whatever is contained in the VMware template he specified as an option of the `create` action.
- The node does not have Puppet installed on it yet.
- The node is not yet configured to function as a CloudWidget application server.

When Tom first configured Puppet, he set up his workstation with the ability to remotely sign certificates. He did this by creating a certifcate/key pair and then modifying the CA's `auth.conf` to allow that certificate to perform authentication tasks. (To find out more about how to do this, see the auth.conf documentation and the REST API guide.)

This allows Tom to use `puppet node init` to complete the process of getting the new node up and running. `Puppet node init` is a wrapper command that will `install` Puppet, `classify` the node, and sign the certificate (`puppet certicate sign` or `puppet cert sign`). "Classifying" the node tells Puppet which configuration groups and classes should be applied to the node. In this case, applying the `cloudwidget_appserv` class configures the node with all the settings, files and database hooks needed to create a fully configured, ready-to-run app server tailored to the CloudWidget environment.

Note: if Tom had not done the prep work needed for remote signing of certificates he could run the `puppet node install`, `puppet node classify` and `puppet cert sign` commands seperately.

Now Tom needs to run Puppet on the new node in order to apply the configuration. He could wait 30 minutes for Puppet to run automatically, but instead he SSH's into the machine and runs Puppet interactively with `puppet agent --test`.

At this point we have:

- A new virtual machine node with Puppet installed
- The node has a signed certificate and is an authorized member of the CloudWidget deployment
- Puppet has fully configured the node with all of the bits and pieces needed to go live and start doing real work as a fully functioning CloudWidget application server.

The CloudWidget infrastructure is now scaled and running at acceptable loads. Tom leans back and takes a sip of his coffee. It's still hot.

- Next: Troubleshooting Cloud Provisioner

Finding Common Errors

Troubleshooting

Missing .fog File or Credentials

If you attempt to provision without creating a `.fog` file or without populating the file with appropriate credentials you'll see the following error:

On VMware:

```
$ 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
```

On Amazon Web Services:

```
$ 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 existing file to resolve this issue.

Note that versions of fog newer than 0.7.2 may not be fully compatible with Cloud Provisioner. This issue is currently being investigated.

Certificate Signing Issues

ACCESSING PUPPET MASTER ENDPOINT

For automatic signing to work, the computer running Cloud Provisioner (i.e. the CP control node) needs to be able to access the puppet master's `certificate_status` REST endpoint. This can be done in the master's auth.conf file as follows:

```
path /certificate_status
method save
auth yes
allow {certname}
```

Note that if the CP control node is on a machine other than the puppet master, it must be able to reach the puppet master over port 8140.

GENERATING PER-USER CERTIFICATES

The CP control node needs to have a certificate that is signed by the puppet master's CA. While it's possible to use an existing certificate (if, say, the control node was or is an agent node), it's preferable to generate a per-user certificate for a clearer, more explicit security policy.

Start by running the following on the control node: `puppet certificate generate {certname} --ca-location remote` Then sign the certificate as usual on the master (`puppet cert sign {certname}`). Lastly, back on the control node again, run:

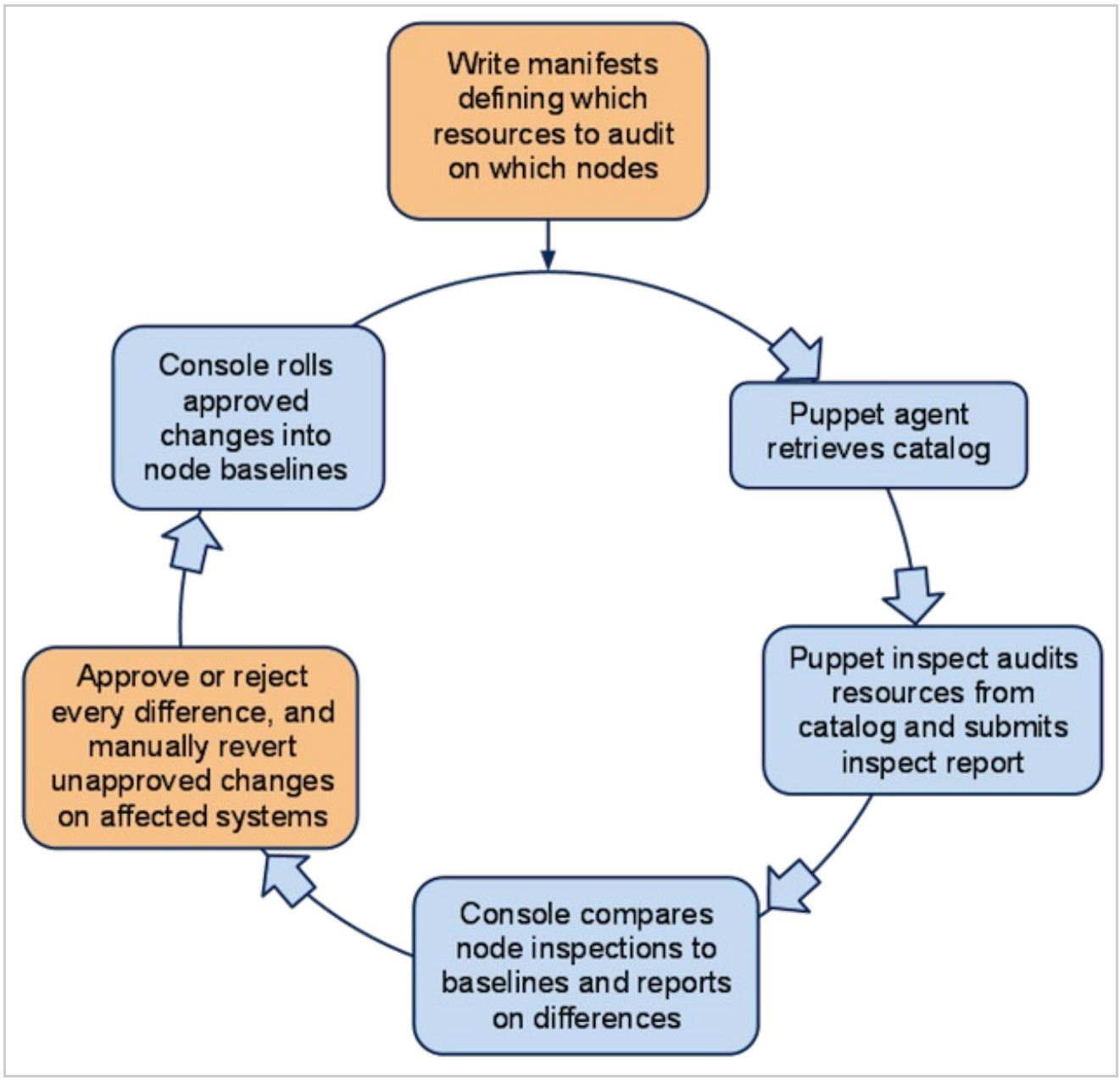
```
  puppet certificate find ca --ca-location remote
  puppet certificate find {certname} --ca-location remote
  This should let you operate under the new certname when you run puppet
commands with the --certname {certname} option.
```

Next: Compliance Basics

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 that define 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.

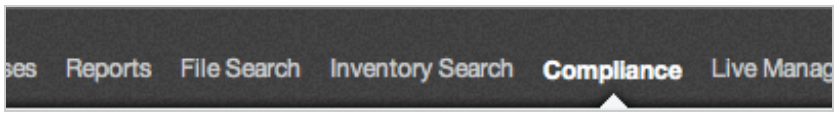
Groups

Although nodes cannot share baselines, nodes in groups can have similar changes approved or rejected in bulk.

The Console's Compliance UI

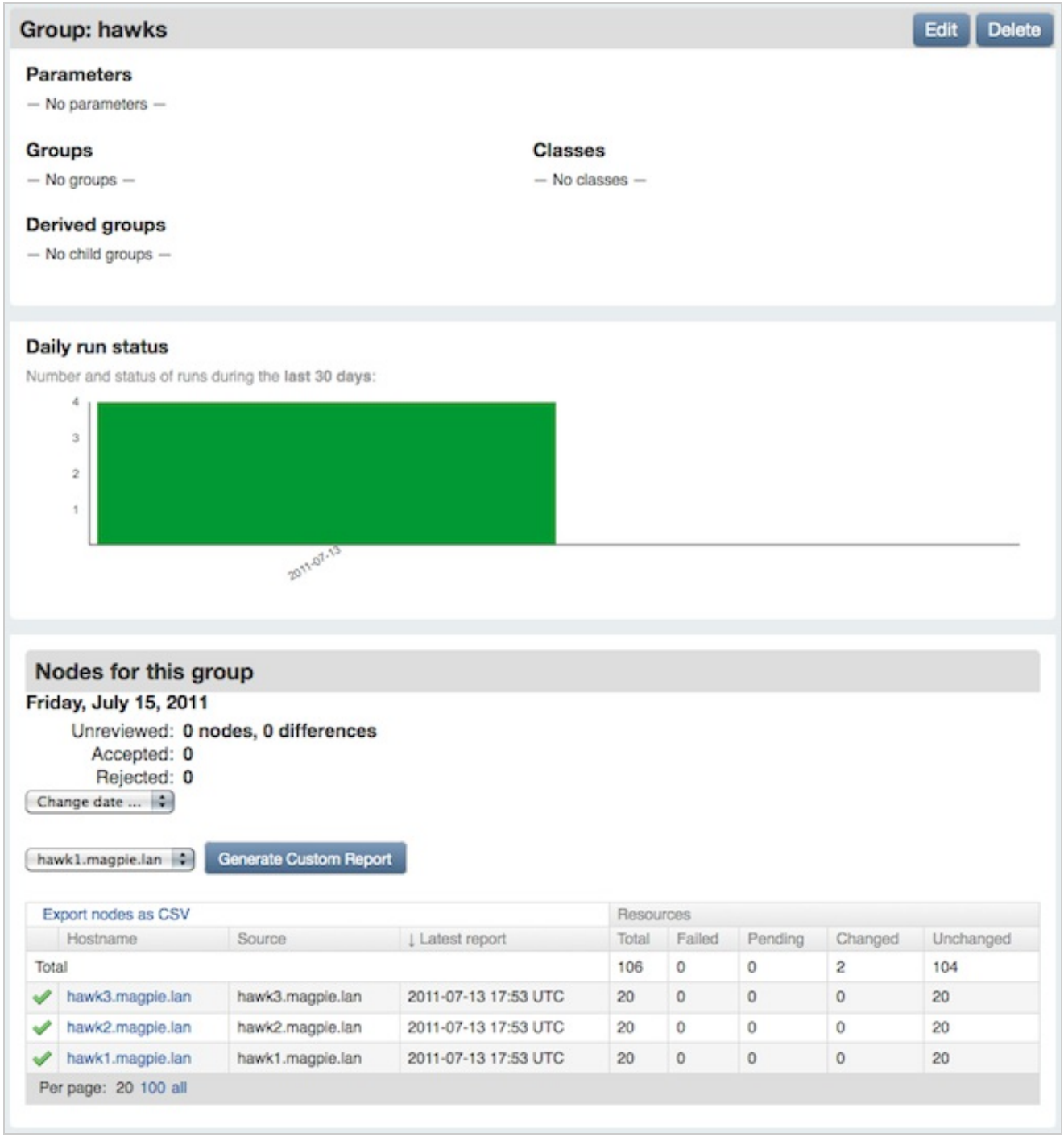
Puppet's compliance UI appears in the console. The compliance module adds the following new elements:

- A 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 main 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.

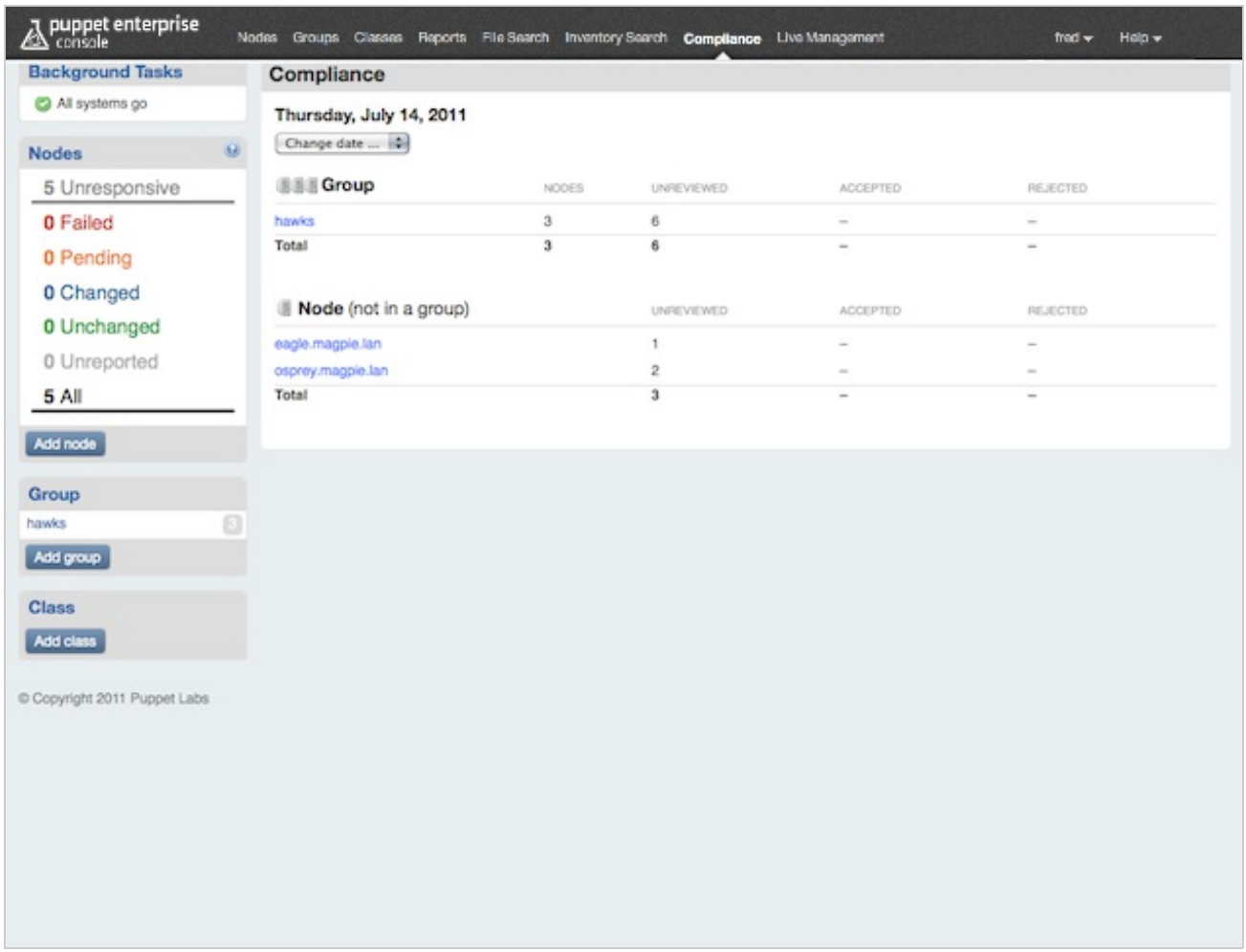
New Controls on Group Pages



Each group page will now have:

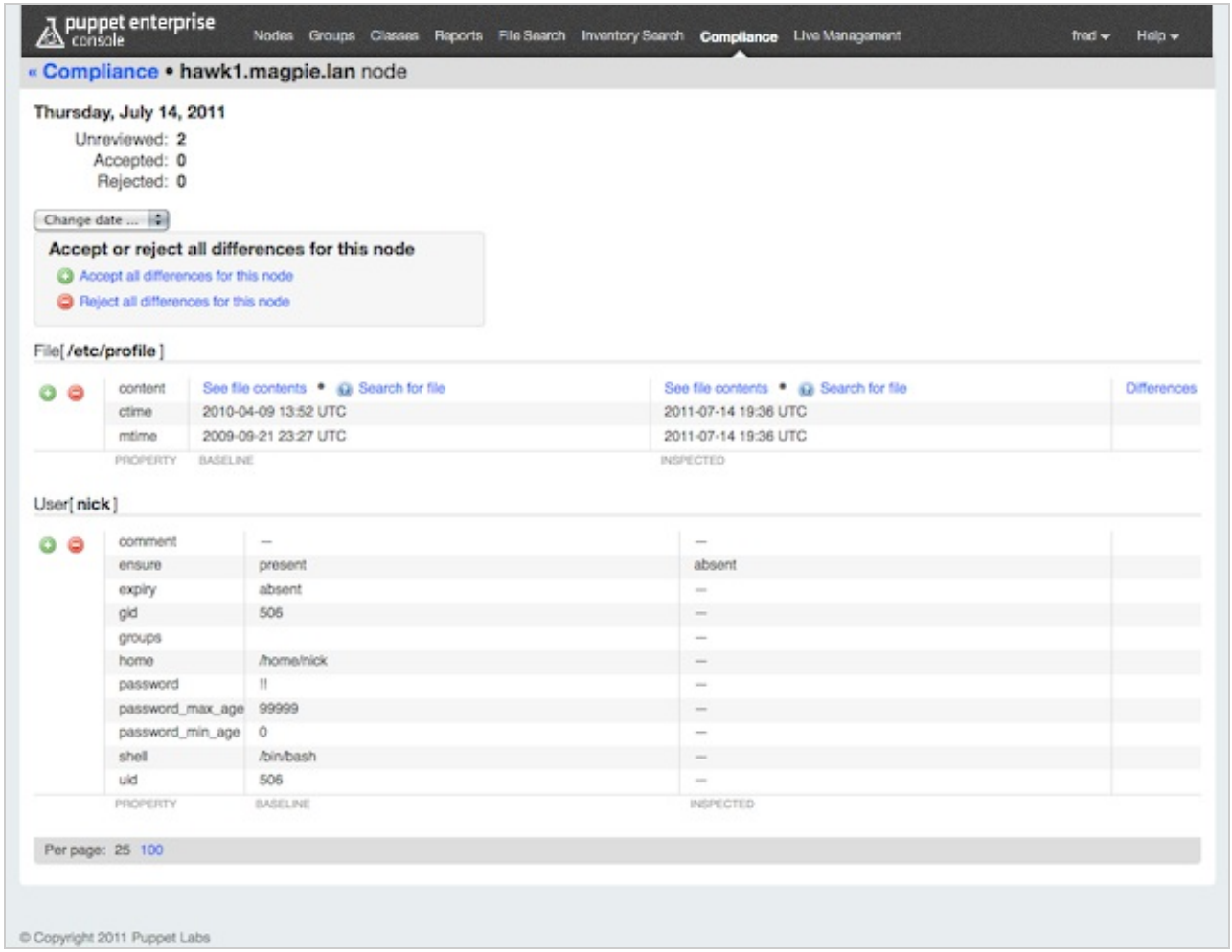
- A compliance summary in its node information section
- A control for generating custom reports

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.

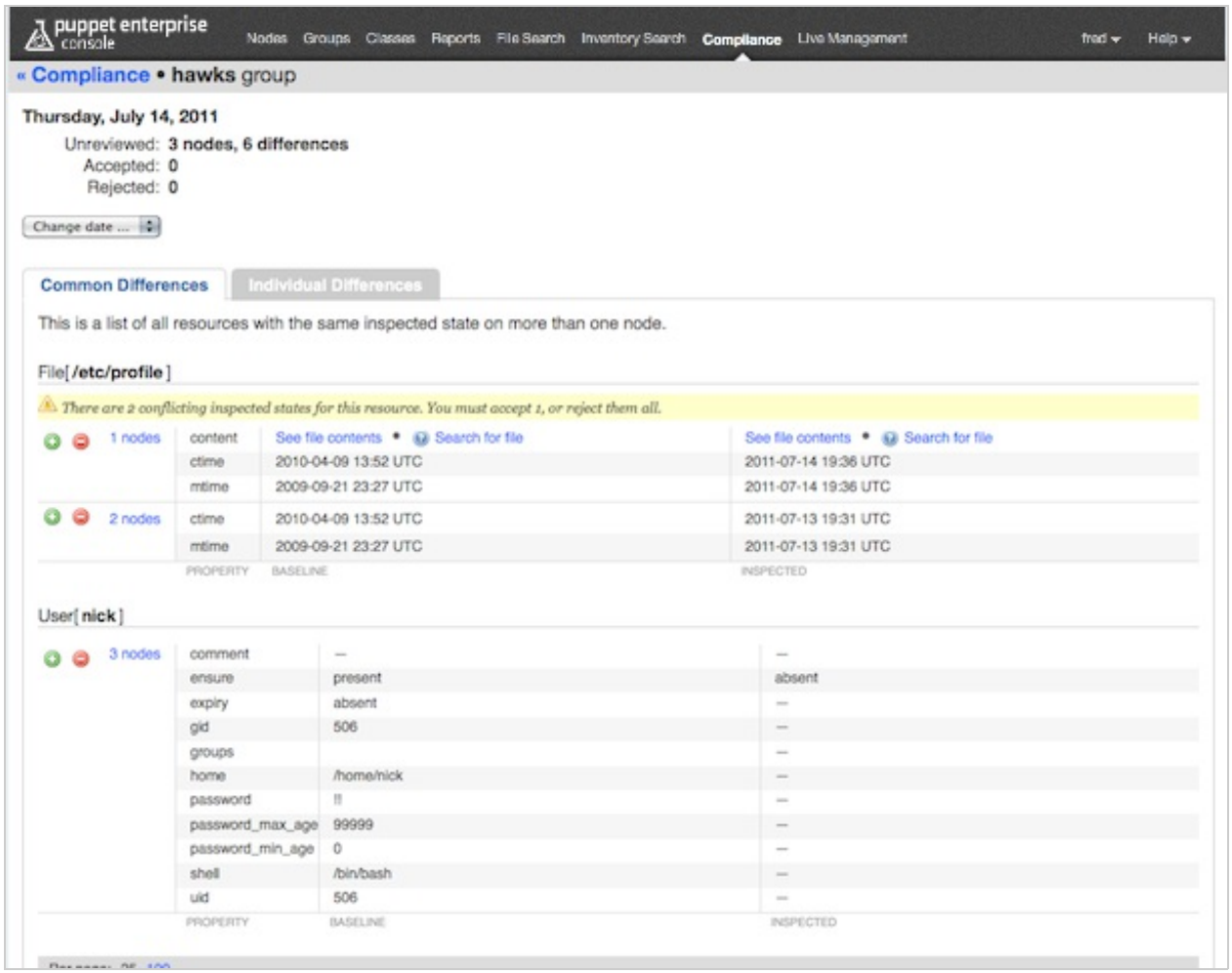
Compliance Node Page

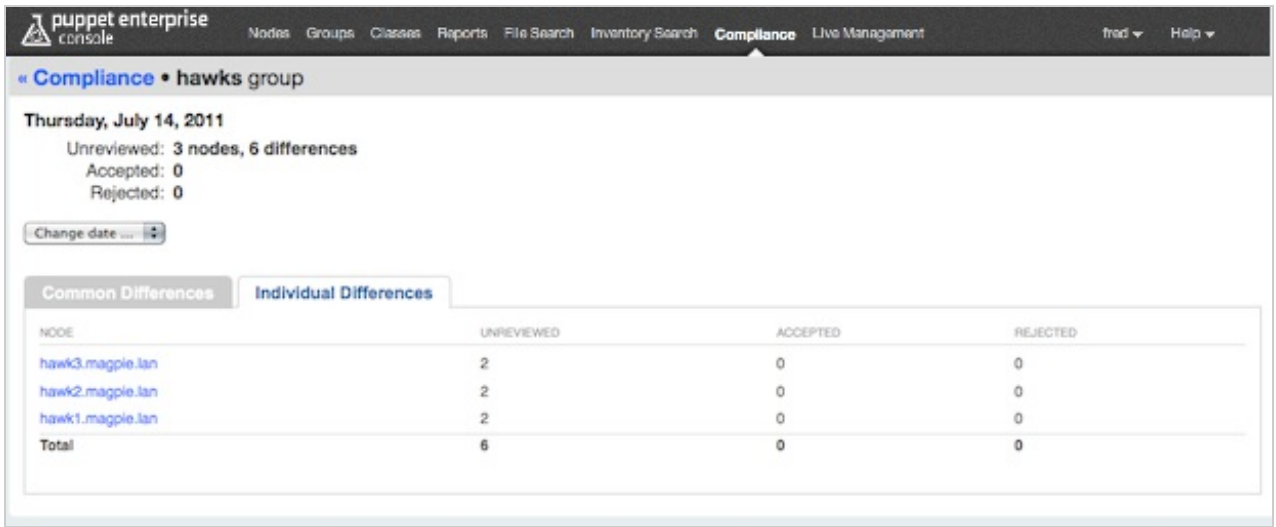


Individual node pages show one node's off-baseline inspection results for a single day. You can accept or reject changes from this page.

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 in bulk
- 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.

- Next: Using the Compliance Workflow

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.

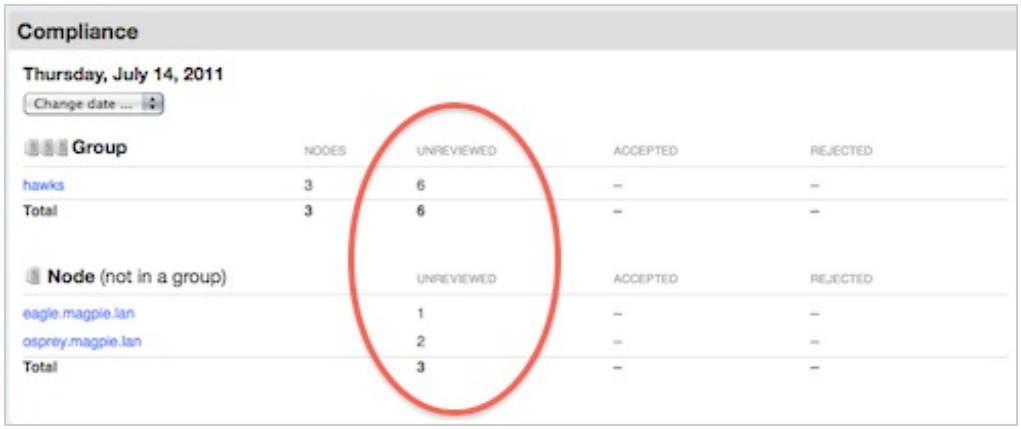
When writing compliance classes in Puppet, you should declare the `audit` metaparameter for each resource. The value of `audit` can be one attribute name, an array of attribute names, or `all`. Puppet can also actively manage attributes of an audited resource.

```
file {'hosts':
  path  => '/etc/hosts',
  audit => 'content',
}
file {'/etc/sudoers':
  audit => [ensure, content, owner, group, mode, type],
}
user {'httpd':
  audit => 'all',
}
```

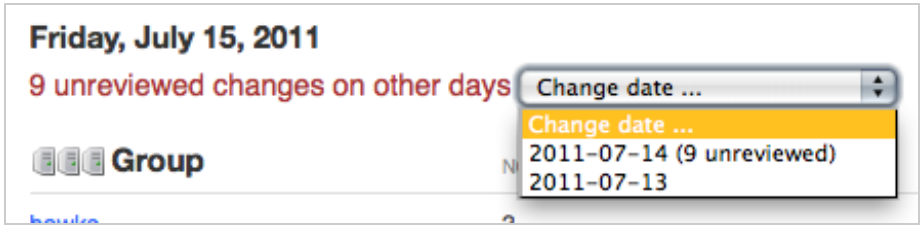
```
    # Allow this user to change their password, but trigger an audit event when
they do:
    user {'admin':
      ensure => present,
      gid    => 'wheel',
      groups => ['admin'],
      shell  => '/bin/zsh',
      audit  => 'password',
    }
```

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

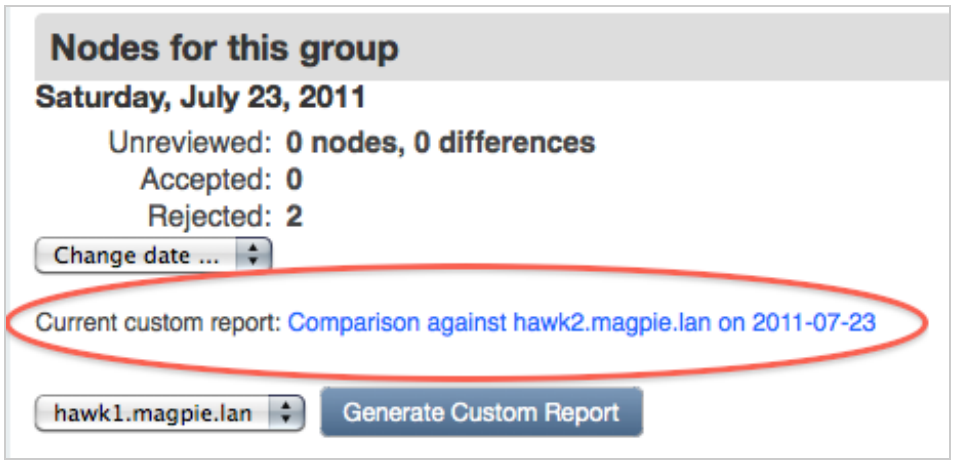
The console can also generate custom reports which compare an entire group to a single member node's baseline. While the day-to-day compliance views are meant for tracking changes to a node or group of nodes over time, custom reports are meant for tracking how far a group of nodes have drifted away from each other.

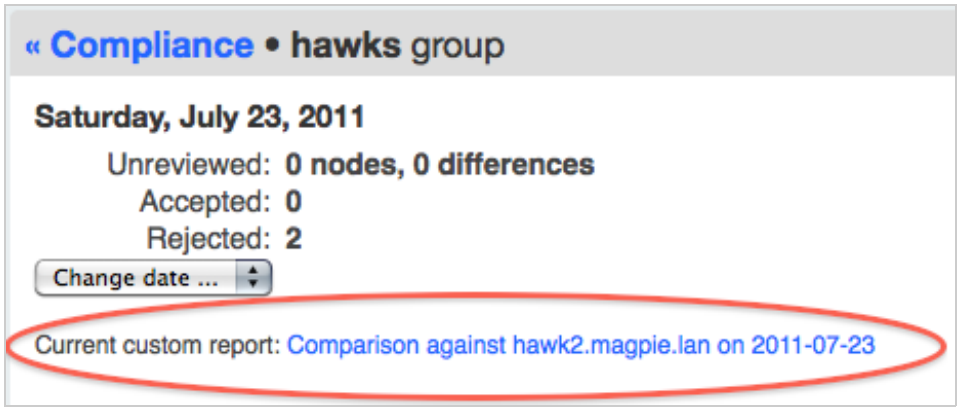
- Custom reports only examine the most recent inspection report for each group member
- Custom reports do not allow you to approve or reject changes

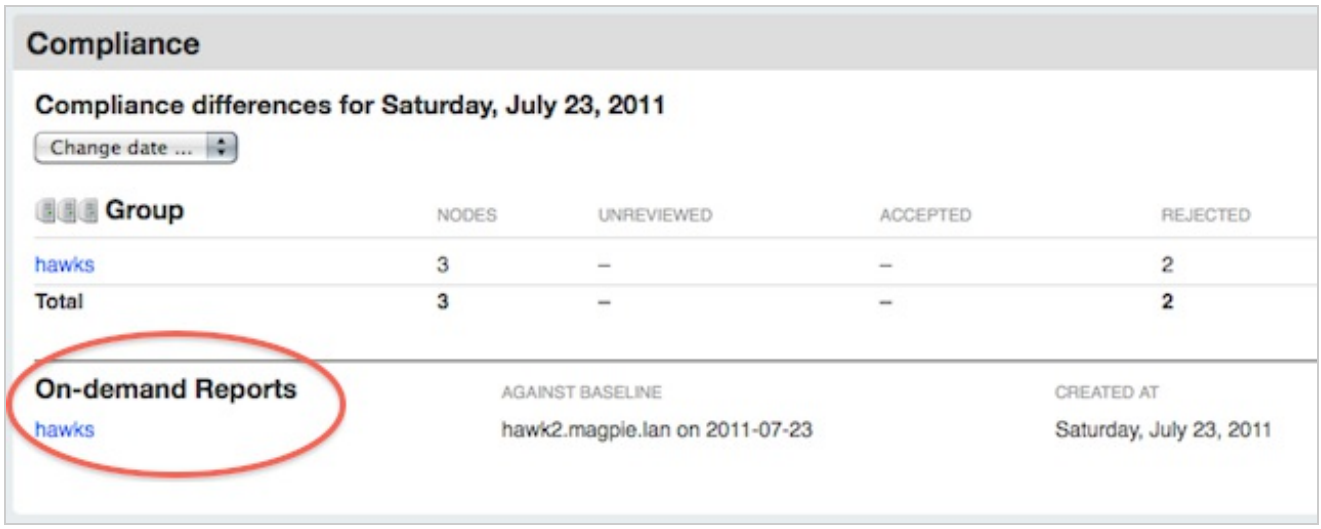
The console will maintain one cached custom report for each group; generating a new report for that group will erase the old one.

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 to update status on the report.)

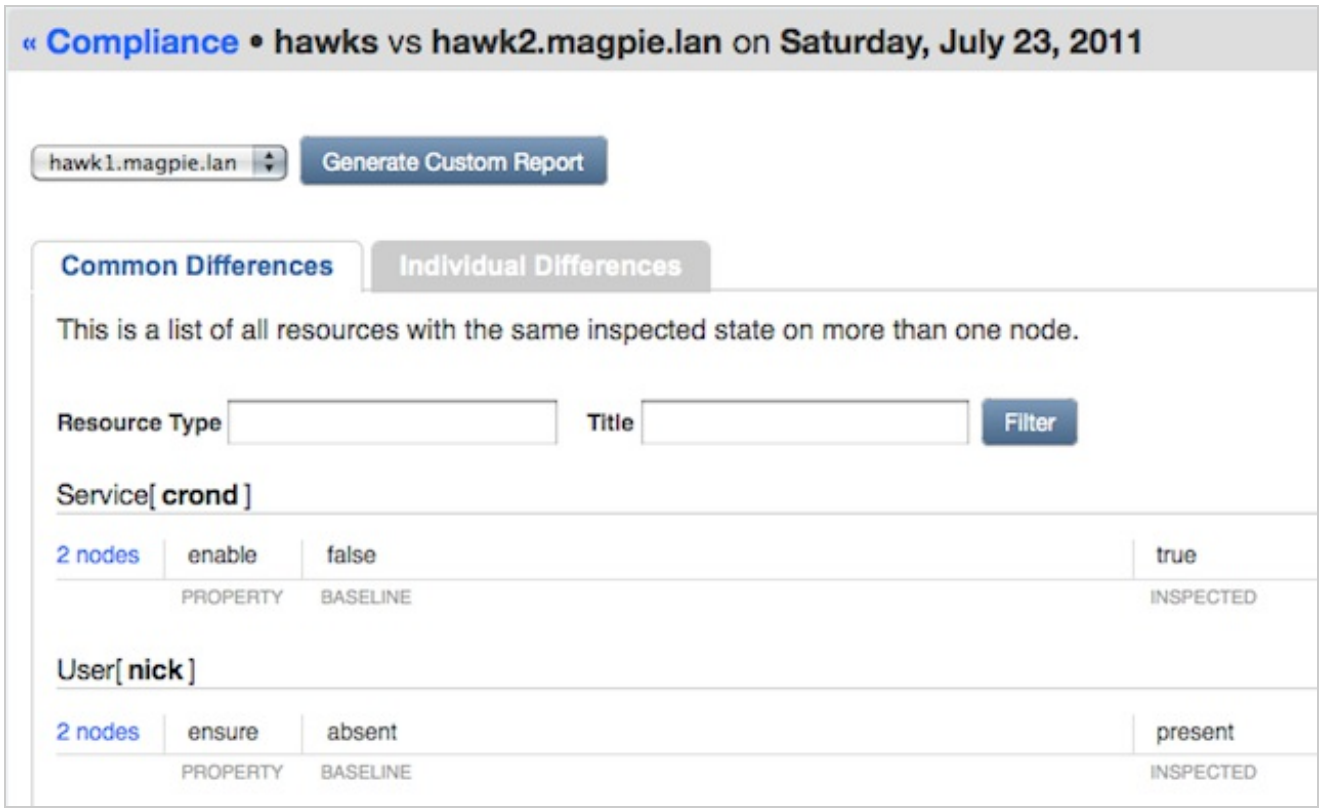
Once generated, custom reports can be viewed from the console's page for that group, the main compliance overview page, and the compliance group pages.







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 can be read in the same fashion. The only difference is that all comparisons are to a single baseline instead of per-node baselines.



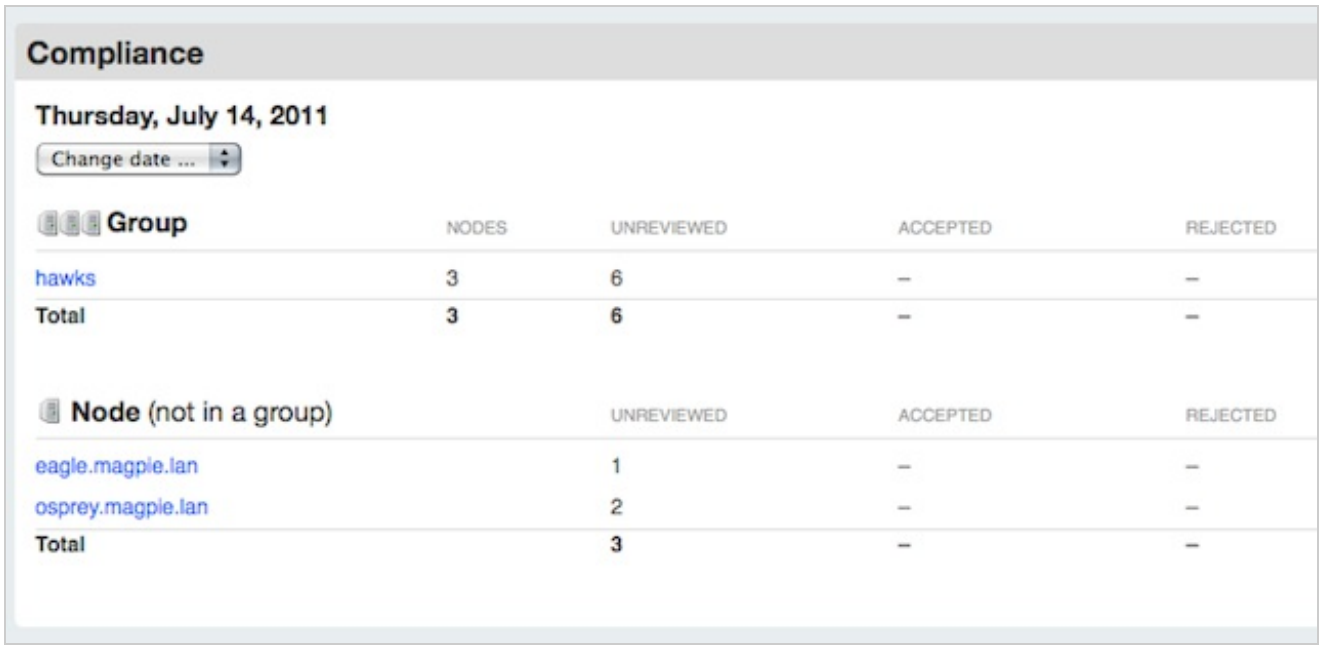
- Next: Compliance Tutorial

Compliance Tutorial

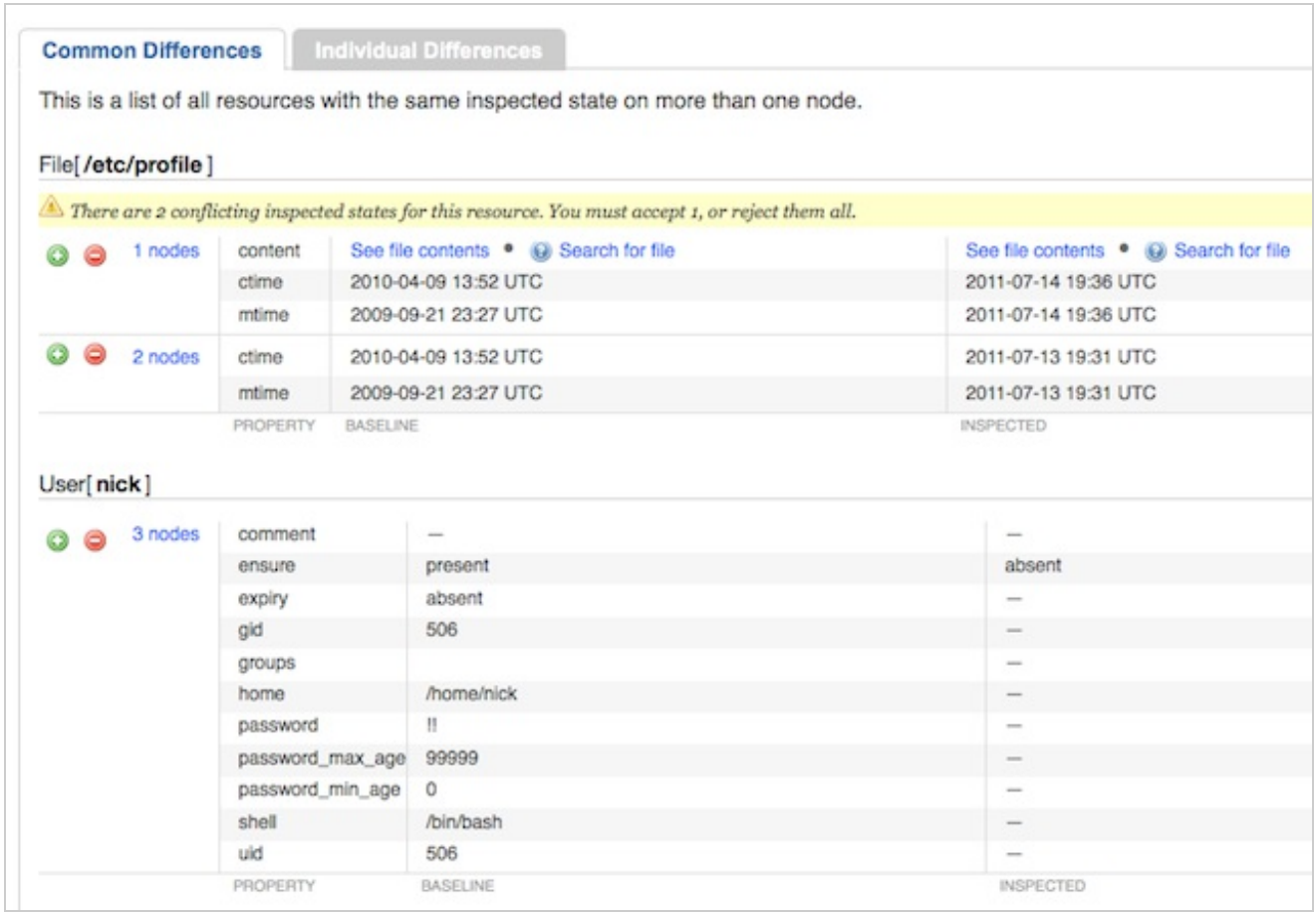
This brief walkthrough shows a compliance workflow auditing the state of the following resources:

- `File['/etc/profile']`
- `File['/etc/syslog.conf']`
- `Service['crond']`
- `Service['sshd']`
- `User['nick']`

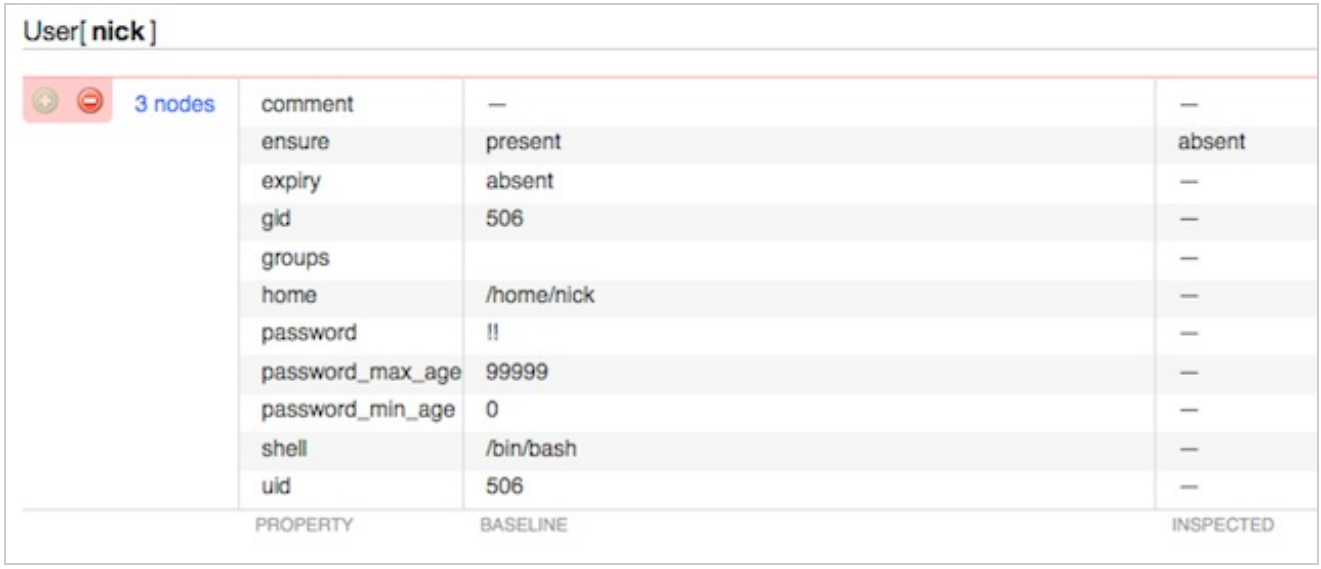
Morning, July 14, 2011



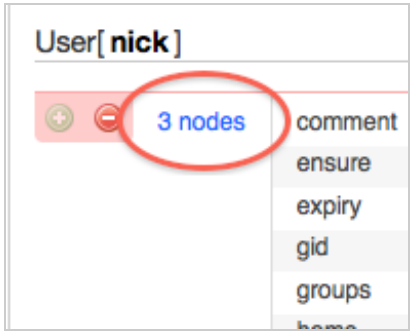
On Thursday morning, the admin notices unreviewed changes in a group of three nodes and a pair of ungrouped nodes. She checks the group first.□



There, she notices that a user was completely deleted from all three nodes, and something odd happened with a file. She immediately clicks the (–) button to reject the deletion of the user…□

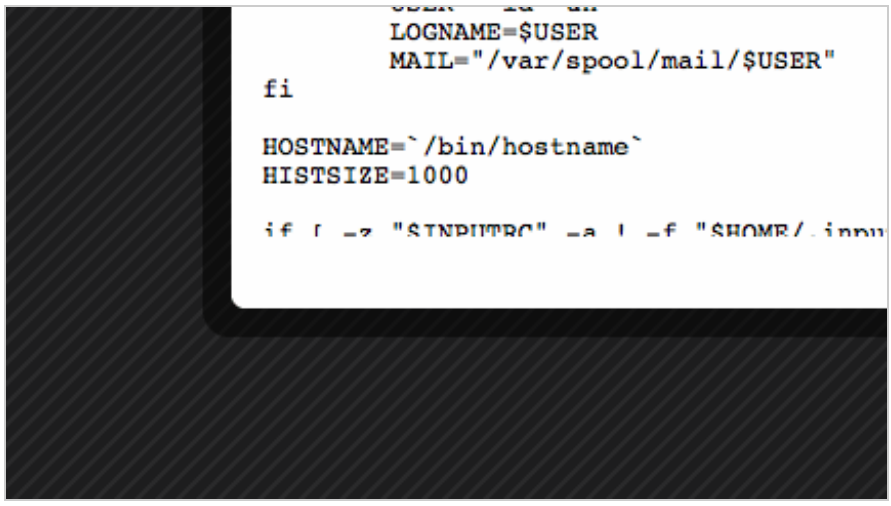


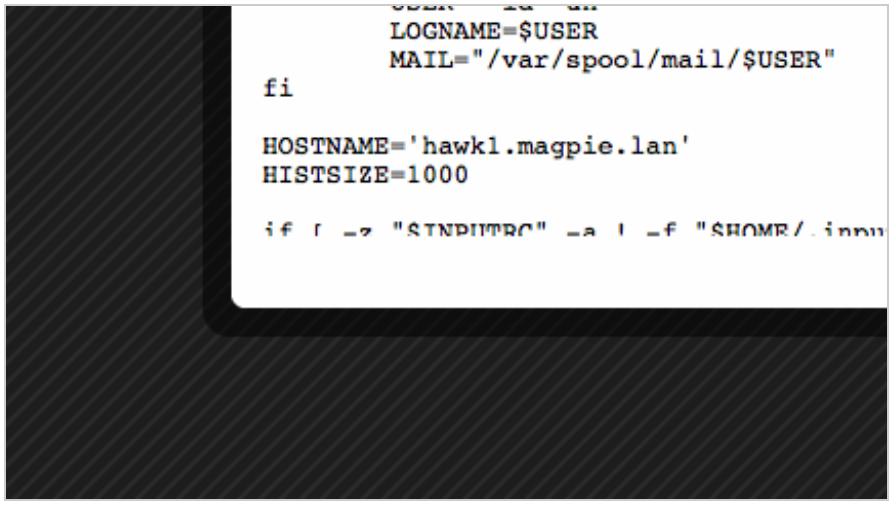
...and manually SSHes to the affected nodes to re-instate the account.




```
[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
...
```

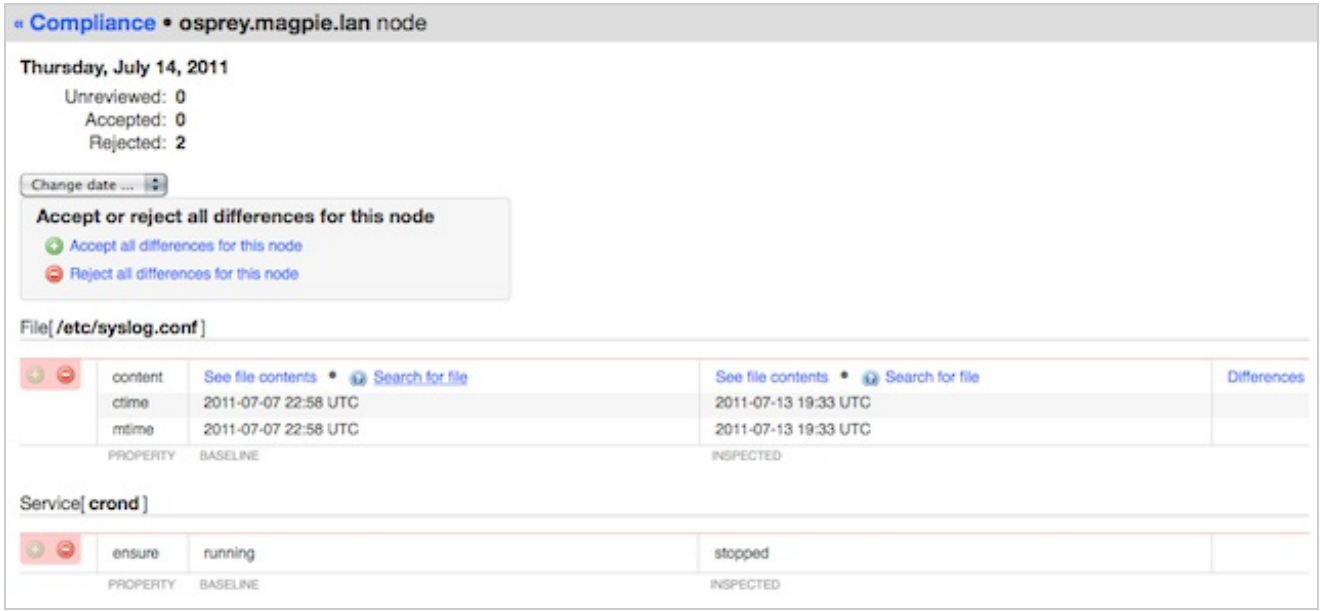
Then she takes a look at the file. It looks like two nodes had the ctime and mtime of the `/etc/profile` file changed, but no edits were made. This was probably nothing, but it piques her interest and she'll ask around about it later. In the meantime, she approves the change, since there's no functional 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.

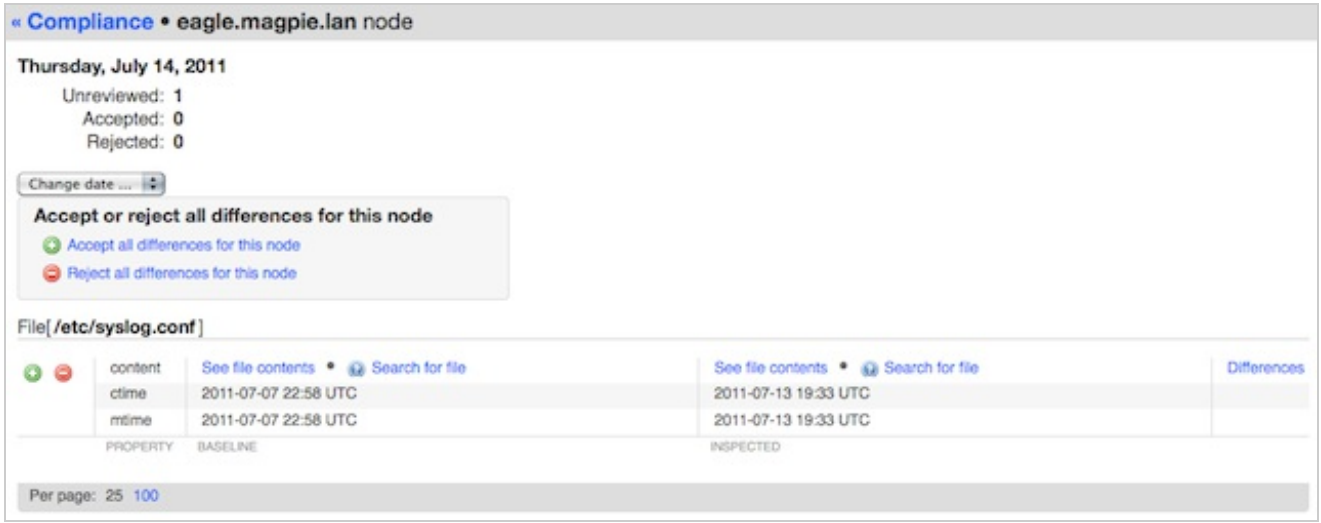
Next, the admin moves on to the individual nodes.

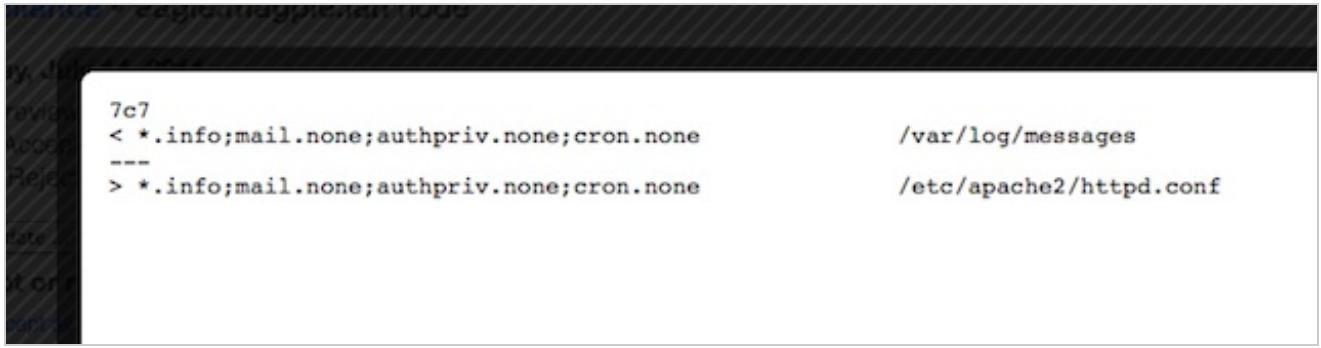


On the osprey server, something has stopped crond, which is definitely not good, and someone has made an edit to `/etc/syslog.conf`. She rejects the cron stoppage and restarts it, then checks the diff on the syslog config:

```
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.

- Next: The `pe_accounts::user` Type

The pe_accounts::user Type

This defined type is part of `pe_accounts`, a pre-built Puppet module that ships with Puppet Enterprise for use in your own manifests.

 NOTE: The `pe_accounts` module is not yet supported on Windows nodes.

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 (except Windows).

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`

Specifies whether the user and its primary group should exist. Valid values are `present` and `absent`. Defaults to `present`. Note that when a user is created, a group with the same name as the user is also created.

`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.

`managehome`

A boolean parameter that dictates whether or not a user's home directory should be managed by the `account` type. If `ensure` is set to `absent` and `managehome` is `true`, the user's home directory will be recursively deleted.

- Next: The `pe_accounts` Class

The pe_accounts Class

This class is part of `pe_accounts`, a pre-built Puppet module included with Puppet Enterprise.

>  NOTE: `pe_accounts` is not yet supported on Windows nodes.

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 and higher, 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 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`

Specifies whether or not 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`

Specifies whether or not 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`

Specifies whether or not 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`

Specifies the 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`

Specifies 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`

Specifies the path to the `sudoers` file on this system. Defaults to `/etc/sudoers`.

- Next: Troubleshooting Common Problems and Solutions

Troubleshooting Common 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.

Are you upgrading from Open Source Puppet?

PE 2.6.0 introduced a bug that could cause upgrade installations to fail if FOSS Puppet had previously been installed. The issue has been fixed in PE 2.6.1 and later. Please be sure you are using the latest PE packages.

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, run the uninstaller script. See the uninstallation instructions in this guide for full details.

After you have run the uninstaller, you can safely run the installer again.

After Upgrading, Some Users Can't Access the Console

Manually Log Out

A bug in 2.6.0 can leave some users with mangled login cookies. Each affected user should type the following URL (replacing the name of your console server) into their browser's location bar to log out, then attempt to load the console again:

```
https://console.example.com/logout
```

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 "name or service 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 see the hostname agents are using to contact the master, run `puppet agent --configprint server`. If this does not return 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 filebucket on 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.

The Console Has Too Many Pending Tasks

The console either does not have enough worker processes, or the worker processes have died and need to be restarted.

- See here to restart the worker processes
- See here to tune the number of worker processes

Console Account Confirmation Emails Have Incorrect Links

This can happen if the console's authentication layer thinks it lives on a hostname that isn't accessible to the rest of the world. The authentication system's hostname is automatically detected during installation, and the installer can sometimes choose an internal-only hostname.

To fix this:

1. Open the `/etc/puppetlabs/console_auth/cas_client_config.yml` file for editing. Locate the `cas_host` line, which is likely commented-out:

   ```
   authentication:

     ## Use this configuration option if the CAS server is on a host different
     ## from the console-auth server.
     # cas_host: console.example.com:443
   ```

   Change its value to contain the public hostname of the console server, including the correct port.

2. Open the `/etc/puppetlabs/console_auth/config.yml` file for editing. Locate the `console_hostname` line:

   ```
   authentication:
     console_hostname: console.example.com
   ```

Change its value if necessary. If you are serving the console on a port other than 443, be sure to add the port. (For example: `console.example.com:3000`)

Troubleshooting Issues on Windows

Refer to the Windows Troubleshooting page for tips and advice to help you resolve common issues when running PE on Windows.

- Next: Appendix

User's Guide Appendix

This page contains additional miscellaneous information about Puppet Enterprise 2.7.

Glossary

For help with Puppet specific terms and language, visit the glossary

For a complete guide to the puppet language, visit the reference manual

Release Notes

PE 2.7.0

The initial release of PE 2.7.

PUPPET CORE PATCHES

Changes to the current version of Puppet's core are documented in the Puppet Release notes. PE 2.7 uses a specially patched version of Puppet 2.7.19. These patches address the following:

- In some cases, an improperly functioning alias between scope and named_scope could cause inventory service to fail. This has been fixed. For details, see Issue 16376.
- The REST API does not return correct metadata for `GET certificate_request/{certname}` or `GET /certificate_status/{certname}`.This has been corrected. For details, see Issue 15731.
- A bug related to the use of hyphens in variable names caused unpredictable behavior when interpolating variables. This has been fixed. There are numerous tickets associated with this issue. See the related issues listed on Issue 10146.
- Release notes for the PE 2.5.x series are available here.

Known Issues

As we discover them, this page will be updated with known issues in Puppet Enterprise 2.7.x. Fixed

issues will be removed from this list and noted above in the release notes. 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 `2.7.19 (Puppet Enterprise 2.7.0)`. To upgrade to a newer version of Puppet Enterprise, see the chapter on upgrading.

The following issues affect the currently shipped version of PE and all prior releases in the 2.x.x series, unless otherwise stated.

Issues with Compliance UI

There are two issues related to incorrect Compliance UI behavior:

- Rejecting a difference by clicking (-) results in an erroneous display (Google Chrome only).
- The user account pull-down menu in the top level compliance tab ceases to function after a host report has been selected.

EC2/Dual-homed Systems Report Incorrect URIs for the Console.

During installation, the PE installer attempts to automatically determine the URI where the console can be reached. On EC2 (and likely all other dual-homed systems), the installer incorrectly selects the internal, non-routable URI. Instead, you should manually enter the correct, external facing URI of the system hosting the console.

Answer file required for some SMTP servers.

Any SMTP server that requires authentication, TLS, or runs over any port other than 25 needs to be explicitly added to an answers file. See the advanced configuration page for details.

Upgrading the Console Server Requires an Increased MySQL Buffer Pool Size

An inadequate default MySQL buffer pool size setting can interfere with upgrades to Puppet Enterprise console servers.

The PE 2.7 upgrader will check for this bad setting. If you are affected, it will warn you and give you a chance to abort the upgrade.

If you see this warning, you should:

- Abort the upgrade.
- Follow these instructions to increase the value of the `innodb_buffer_pool_size` setting.
- Re-run the upgrade.

If you have attempted to upgrade your console server without following these instructions, it is possible for the upgrade to fail. 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
================================================================================
```

If you have suffered a failed upgrade, you can fix it by doing the following:□

- 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;
```

- Follow the instructions for increasing the `innodb_buffer_pool_size` and restart the MySQL server.
- Re-run the upgrader, which should now finish successfully.

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

`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
```

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.

To improve the display of Puppet man pages, you can use your system `gem` command to install the `ronn` gem:

```
$ sudo gem install ronn
```