Puppet Dashboard Manual

(Generated on July 01, 2013, from git revision 46784ac1656bd7b57fcfb51d0865ec7ff65533d9)

Puppet Dashboard 1.2 Manual

This is the manual for Puppet Dashboard 1.2.

Overview

Puppet Dashboard is a web interface for Puppet. It can view and analyze Puppet reports, assign Puppet classes and parameters to nodes, and view inventory data and backed-up file contents.

Chapters

Installing Puppet Dashboard

This is a chapter of the Puppet Dashboard 1.2 manual.

NAVIGATION

Overview

Puppet Dashboard is a Ruby on Rails web app that interfaces with Puppet. It will run on most modern Unix-like OSes (including Mac OS X and most Linux distributions), requires a certain amount of supporting infrastructure, and can be deployed and served in a variety of ways. Dashboard's web interface supports the following browsers:

- Chrome (current versions)
- Firefox 3.5 and higher

- Safari 4 and higher
- Internet Explorer 8 and higher

Installing, in Summary

In outline, the steps to get Dashboard running are:

- Installing the external dependencies
- Installing the Dashboard code
- Configuring Dashboard
- Creating and configuring a MySQL database
- Testing that Dashboard is working
- Configuring Puppet
- Starting the delayed job worker processes
- Running Dashboard in a production-quality server

After completing these tasks, Dashboard's main functionality will be on-line and working smoothly. You can then configure Dashboard further and enable optional features

If you are trying to upgrade Puppet Dashboard instead of installing it from scratch, see the chapter of this manual on upgrading instead of reading further in this chapter. If you're looking for a vastly simplified installation and maintenance process, download a free trial of Puppet Enterprise and try its improved and extended web console.

Installing Dependencies

Dashboard is distributed with the version of Rails it uses, as well as most of its other dependencies, but you'll have to ensure that the following software is installed:

- RubyGems
- Rake version 0.8.3 or newer
- MySQL database server version 5.x
- Ruby-MySQL bindings version 2.7.x or 2.8.x

On most OSes, you'll be able to install all of these easily with the OS's package tools.

Note: Puppet supplies Ruby 1.8.7 packages for Enterprise Linux 5 and its variants in order meet the Ruby versioning requirement for Dashboard. Also note, these packages replace existing Ruby packages in your system.

INSTALLING DEPENDENCIES UNDER UBUNTU 10.04 LTS

Due to issues with Ubuntu 10.04 LTS's version of Ruby, you can install most dependencies from packages but must manually build `gem`. Additionally, if you encounter performance issues, you may wish to manually upgrade your version of Ruby to patch level 299 or higher.

These instructions assume a fresh install of the OS, and may differ depending on its current configuration. The commands must be run from one of the standard shells, preferably `bash`, `dash`, or `zsh`.

1. Install the operating system packages:

```
apt-get install -y build-essential irb libmysql-ruby libmysqlclient-dev \
  libopenssl-ruby libreadline-ruby mysql-server rake rdoc ri ruby ruby-dev
```

2. Install the RubyGems package system, using the following shell script — do not use the `rubygems` packaged with the operating system:

```
(
  URL="http://production.cf.rubygems.org/rubygems/rubygems-1.3.7.tgz"
  PACKAGE=$(echo $URL | sed "s/\.[^\.]*$//; s/^.*\///")

  cd $(mktemp -d /tmp/install_rubygems.XXXXXXXXXX) && \
  wget -c -t10 -T20 -q $URL && \
  tar xfz $PACKAGE.tgz && \
  cd $PACKAGE && \
  sudo ruby setup.rb
)
```

3. Create `gem` as an alternative name for the `gem1.8` command:

```
update-alternatives --install /usr/bin/gem gem /usr/bin/gem1.8 1
```

Installing Puppet Dashboard

Your three options for installing Dashboard are to use the Puppet Labs package repositories, install the source from Git, or download a tarball of the source. Whichever way, you'll end up with a single directory — as Rails apps are self-contained, all of Dashboard's code, configuration, and run data will be stored in the same area. Any rake tasks mentioned later in this manual should be performed from a shell in this main directory, and any relative paths will refer to locations within it.

Installing from Packages

The best way to install Dashboard is with Puppet Labs' official package repositories. This will automatically handle Dashboard's dependencies, and will make for easier upgrades when new versions are released.

When installing from packages, Dashboard will be installed in `/usr/share/puppet-dashboard`, and the `puppet-dashboard` user and group will own the files; this user will be automatically created if it doesn't exist.

ENABLING THE REPOSITORY

Before installing, follow the instructions here to enable the Puppet Labs package repository for your system.

INSTALLING DASHBOARD

On RPM-based systems, install Puppet Dashboard via Yum:

```
$ sudo yum install puppet-dashboard
```

On Debian-based systems, install Puppet Dashboard via Apt:

```
$ sudo apt-get update
$ sudo apt-get install puppet-dashboard
```

Installing from Git

If you're unable to use the Dashboard packages on your system, the next best way to install Dashboard is from the Puppet Labs Git repo. In the directory where you want Dashboard installed (we suggest `/opt/` or `/usr/share/`), run:

```
git clone git://github.com/puppetlabs/puppet-dashboard.git
cd puppet-dashboard
git checkout v1.2.0
```

The advantage of using Git is that you have an easier upgrade path: you don't have to manually remember which files to preserve when upgrading, and the actual process of upgrading is much simpler. The disadvantage is that it basically turns Git into another dependency. See upgrading for more details.

If you haven't installed Dashboard from a package, you must create a user and group for Dashboard and chown all its files to be owned by that user and group:

```
sudo chown -R puppet-dashboard:puppet-dashboard /opt/puppet-dashboard
```

Installing from Source Tarballs

If you prefer not to use Git, you can simply download the most recent release of Puppet Dashboard and extract it into your install location:

```
wget http://puppetlabs.com/downloads/dashboard/puppet-dashboard-1.2.0.tar.gz
tar -xzvf puppet-dashboard-1.2.0.tar.gz
mv puppet-dashboard-1.1.1 /opt/puppet-dashboard
```

If you haven't installed Dashboard from a package, you must create a user and group for Dashboard and chown all its files to be owned by that user and group.

```
sudo chown -R puppet-dashboard:puppet-dashboard /opt/puppet-dashboard
```

Configuring Dashboard

Dashboard needs a `config/database.yml` file and a `config/settings.yml` file. It ships with functional examples of each, as `config/database.yml.example` and `config/settings.yml.example` respectively.

`database.yml`

The database settings file is located at `config/database.yml`, and an example file can be found at `config/database.yml.example`. This file should be a YAML hash with one key for each of the standard Rails environments: production, development, and test.

- The "production" environment gives the best performance, and should be used most of the time by most users. Rails does not consider production its default environment, and you must specify it manually with the `RAILS_ENV` environment variable when running any rake tasks or starting a WEBrick server.
- The "development" environment gives worse performance, but yields better logging and error reporting when something goes wrong.
- The "test" environment is only used for running Dashboard's automated tests, and should never be used by most users.

You will likely only ever be using the production environment. You may wish to use the same database for the production and development environments, as this can remove the pain of having to specify `RAILS_ENV=production` for every rake task and gives you the option of running a temporary instance with the same data in the development environment (if you ever need to collect detailed error messages, for example). You should not use the same database for the test environment, as it gets destroyed every time the tests are run.

Each environment in the `database.yml` file should be a hash with keys for `database`, `username`, `password`, `encoding`, and `adapter`. At the moment, `adapter` can only be `mysql`, and `encoding` should always be `utf8`.

Do not give Dashboard the keys to MySQL's `root` user account — create a new database user for it (preferably named "dashboard") and choose a secure password.

Since the `database.yml` file has to contain Dashboard's database user and password in cleartext, you should set its mode to 660 (or some other world-unreadable mode). If you've created the file while logged in as a normal user, be sure to chown it to the Dashboard user.

`settings.yml`

The general settings file should be a YAML hash. When first configuring Dashboard, you should simply make a copy of `settings.yml.example`, as it's unlikely that you'll need to change any settings yet. (Although you may wish to change `date_format`, `custom_logo_url`, or `no_longer_reporting_cutoff`.)

Creating and Configuring a MySQL Database

Dashboard needs a user, a password, and at least one database. Create these using the names and passwords you chose in your `database.yml` file.

This will require that you use some external MySQL administration utility;[1] in the standard command-line `mysql` client, the commands will look something like this:

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

(See the MySQL manual for more information.)

Tuning

After creating the database and user, you'll need to configure MySQL's maximum packet size to permit larger rows in the database. On rare occasions, Dashboard can send up to 17MB of data in a single row, and to ensure that it will function under load, you should edit `/etc/mysql/my.cnf` to increase the limit to at least 24MB (we recommend 32MB or more):

```
# Allowing 32MB allows an occasional 17MB row with plenty of spare room
max_allowed_packet = 32M
```

Either restart the MySQL server for this setting to take effect, or run the following command from the `mysql` client:

```
mysql> set max_allowed_packet = 33554432;
```

Preparing Schema

Once Dashboard has its database, it can create its tables, but this has to be done manually with the `db:migrate` rake task. For typical use with the `production` environment:

```
rake RAILS_ENV=production db:migrate
```

For developing the software using the `development` and `test` environments:

```
rake db:migrate db:test:prepare
```

The `db:migrate` task can be safely run multiple times.

Testing That Dashboard is Working

You can now run Dashboard using Ruby's built-in WEBrick server:

```
sudo -u puppet-dashboard ./script/server -e production
```

This will start a Dashboard instance on port 3000 using the "production" environment. You'll be able to view Dashboard's UI in a web browser at http://dashboardserver:3000, and your puppet master can now be configured to use it for reporting and node classification. Note that:

- You will need to have already created the puppet-dashboard user and group.
- You must specify the environment manually if you're using anything other than "development".

Running under WEBrick isn't feasible for production use, since it can't handle concurrent requests, but it can be useful when first getting Dashboard and Puppet configured. If you'd rather skip straight to a production-ready deployment, see the relevant section below.

Configuring Puppet

Puppet Dashboard has two main functions: report viewer/analyzer, and node classifier. Puppet can use either of these functions or both of them. Once you have puppet configured, you'll need to restart puppet master.

Using Dashboard for Reports

For Dashboard to receive reports, there are two requirements:

- All agent nodes have to be configured to submit reports to the master.
- The master has to be configured to send reports to Dashboard.

CONFIGURING REPORTS ON PUPPET 2.6.0 AND NEWER

- Make sure that all agents have reporting turned on:

```
# puppet.conf (on each agent)
[agent]
  report = true
```

- Add the `http` report handler to your puppet master's `reports` setting and set `reporturl` to your

Dashboard instance's `reports/upload` URL:

```
# puppet.conf (on puppet master)
[master]
  reports = store, http
  reporturl = http://dashboard.example.com:3000/reports/upload
```

CONFIGURING REPORTS ON PUPPET 0.25.X

Puppet 0.25.x lacks the `http` report handler, so you'll need to do a few extra steps.

- Make sure that all agents have reporting turned on:

```
# puppet.conf (on each agent)
[puppetd]
  report = true
```

- Copy Dashboard's custom report handler into puppet master's `libdir`:

```
# mkdir -p $(puppetmasterd --configprint libdir)/puppet/reports
# cp ext/puppet/puppet_dashboard.rb $(puppetmasterd --configprint
libdir)/puppet/reports/
```

  If the puppet master is a different machine, you'll need to SCP the file to it first. Also note that you may need to edit the report processor if you're running Dashboard on a different server or port, as it assumes Dashboard is running on localhost:3000.

- Add `puppet_dashboard` to your puppet master's `reports` setting:

```
# puppet.conf (on puppet master)
[puppetmasterd]
  reports = store, puppet_dashboard
```

- If your puppet master server is also running puppetd AND it has `pluginsync` turned on, you'll also need to change the agent's `libdir`:

```
# puppet.conf (on puppet master)
[puppetd]
  pluginsync = true
  libdir = $vardir/agent_lib
```

Using Dashboard for Node Classification

You can use Dashboard's external node classifier (ENC) alongside traditional Puppet DSL node definitions. However, if you use your own custom ENC (or LDAP nodes), you won't be able to use

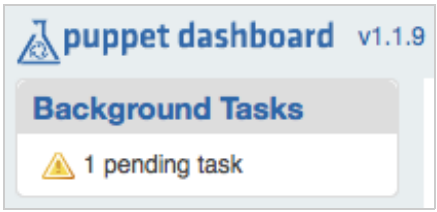
Dashboard's ENC.

To use Dashboard's ENC, you'll need to set the puppet master's `node_terminus` and `external_nodes` settings, and make sure the master can access Dashboard's node classification script, which is located at `bin/external_nodes`. This script's behavior can be overridden by setting environment variables; unless you're serving Dashboard over HTTPS, the only one you'll need to set is `PUPPET_DASHBOARD_URL`.

```
# puppet.conf (on puppet master)
[master]
  node_terminus = exec
  external_nodes = /usr/bin/env PUPPET_DASHBOARD_URL=http://localhost:3000
/opt/puppet-dashboard/bin/external_node
```

Testing Puppet's Connection to Dashboard

After restarting puppet master, you can run one of your puppet agents with `puppet agent --test` to check whether the configuration is correct. The agent should be able to retrieve its catalog and complete its run, and when you reload the Dashboard UI in your web browser, you should see "1 pending task" under the "Background Tasks" heading in the upper left corner.



This means the report arrived as expected, and will be processed once the delayed job workers are active.

Starting and Managing Delayed Job Workers

Dashboard uses a `delayed_job` queue to asynchronously process resource-intensive tasks. Although Dashboard 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 Dashboard's UI.

A future version of Dashboard will ship with init scripts which will let you manage the workers with Puppet or your platform's service tools, but in the meantime, you must either use the provided monitor script or start non-daemonized workers individually with the provided rake task.

Using the monitor script

Dashboard ships 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:

```
$ sudo -u puppet-dashboard 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.

MONITORING THE MONITOR

For additional reliability, you might want to use a standard service monitoring tool like god, monit, or runit to supervise the `script/delayed_job` monitor. You can also look into other ways to run `delayed_job` workers, as it's becoming a fairly standard component in the Rails world.

Using the `jobs:work` Rake Task

We don't recommend using rake-started workers for production, but they can be useful when testing or developing Dashboard. To start a single worker in the production environment:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production jobs:work
```

Running Dashboard in a Production-Quality Server

Although you may have tested Dashboard using the included WEBrick server script, you'll need to deploy in a production-quality server like Apache with Passenger or Nginx with Passenger, Thin, or Unicorn before rolling out Dashboard to your entire site. Instructions follow for running Dashboard under Apache with Passenger, but as Dashboard is a fairly standard Rails application, instructions for using any popular backend should be easily adaptable.

Serving Dashboard With Passenger and Apache

First, you'll need to ensure that Apache 2.2 and Phusion Passenger are installed. The Passenger website has installation instructions, but it's quite possible that your OS vendor has already packaged Passenger, which can make for a much easier install.

Once Passenger is enabled, copy Dashboard's example vhost from `ext/passenger/dashboard-vhost.conf` into Apache's `sites-enabled` directory and edit it to match your Dashboard installation. Passenger runs Rails apps in the production environment by default, so you won't need to explicitly set the environment (with the `RailsEnv` directive in the vhost configuration) unless you plan to run it in development mode. The parts of the file you'll need to edit are:

- The port on which to serve Dashboard. This defaults to 80, but if you want to serve it on Puppet's preferred port of 3000, you'll need to change the opening tag of the vhost definition block to `<VirtualHost *:3000>` and insert a `Listen 3000` directive above it.
- The subdomain you'll be serving Dashboard from, which is generally just the fully-qualified

domain name of this machine. Put this in the `ServerName` directive.

- The location of Dashboard's `public` directory, which should go in both the `DocumentRoot` directive and the `<Directory>` block opening tag.
- Your preferred log file locations, which go in the `ErrorLog` and `CustomLog` directives.
- The paths to Passenger, `mod_passenger`, and Ruby. But before you tweak these, scan the rest of Apache's config files: if you installed Passenger from a vendor package, it probably already inserted a global config file to make sure it's loaded, in which case you can safely comment out the first three lines of this vhost config. Otherwise, point the `LoadModule`, `PassengerRoot`, and `PassengerRuby` directives at the top of the file to the correct files and directories.

If you prefer to roll your own vhost config, see the Passenger user's guide and note that:

- The `DocumentRoot` should point to Dashboard's `public` directory, which needs to allow all access and have the `MultiViews` option turned off.
- Passenger will need either the per-server `RailsAutoDetect` directive set to `On` (which is its default state), or a `RailsBaseURI` directive in the vhost definition.

Once Dashboard's vhost config is properly configured, simply restart Apache and test that Puppet can communicate with Dashboard, as described above.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

1. Instead of creating a database manually, you can also use the `db:create` or `db:create:all` tasks, but these require that Dashboard's MySQL user already exist and have the appropriate permissions on the requested database. Since you'll likely need to use raw SQL commands or another external tool to do that, you might as well just create the databases while you're in there.↩

Upgrading Puppet Dashboard

This is a chapter of the Puppet Dashboard 1.2 manual.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard

- Maintaining Dashboard
- Using Dashboard
- Rake API

Overview

Upgrading Dashboard from a previous version generally consists of the following:

- Stopping the webserver and delayed jobs workers
- Upgrading the Dashboard code itself
- Running any new database migrations
- Restarting the webserver and delayed jobs workers

In addition, there are several tasks you must take into account when upgrading from certain versions.

- Upgrading from pre-1.2 versions
- Upgrading from pre-1.1 versions

Note that all rake tasks should be performed from a shell in the directory that contains Dashboard's code. Any relative paths mentioned below refer to locations within this directory. If you are running Dashboard in the recommended "production" environment, note that Rails does not consider production its default environment, and you must specify it manually with the `RAILS_ENV` environment variable when running any rake tasks.

Upgrading Code

From Packages

Dashboard installations that used Puppet Labs' packages are the easiest to upgrade. If you installed Dashboard with Yum:

```
$ sudo yum update puppet-dashboard
```

If you installed it with APT:

```
$ sudo apt-get update
$ sudo apt-get install puppet-dashboard
```

If you installed it from an RPM package file:

```
$ sudo rpm -Uvh puppet-dashboard-1.2.0.noarch.rpm
```

If you installed it from a Deb package file:

```
$ sudo dpkg -i puppet-dashboard-1.2.0_all.deb
```

From Git

Upgrading from Git is relatively straightforward, although you will have to re-chown all of Dashboard's files after performing the upgrade.

First, fetch data from the remote repository:

```
$ git fetch origin
```

Before checking out the new release, make sure that you haven't made any changes that would be overwritten:

```
$ git status
```

Dashboard's `.gitignore` file should ensure that your configuration files, certificates, temp files, and logs will be untouched by the upgrade, but if the status command shows any new or modified files, you'll need to preserve them. You could just copy them outside the directory, but the easiest path is to use git stash:

```
$ git add {list of modified files}
$ git stash save "Modified files prior to 1.2.0 upgrade"
```

After that, you're clear to upgrade:

```
$ git checkout v1.2.0
```

(And if you had to stash any edits, you can now apply them:

```
$ git stash apply
```

If they don't apply cleanly, you can abort the commit with `git reset --hard HEAD`, or read up on how to resolve Git merge conflicts.)

Finally, re-chown all Dashboard files to the puppet-dashboard user:

```
$ sudo chown -R puppet-dashboard:puppet-dashboard ./*
```

From Tarballs

If you originally installed Dashboard from a source tarball, you'll need to either pick out all of your modified or created files and transplant them into the new installation, or convert your installation to Git; either way, you should back up the entire installation first.

To convert an existing Dashboard installation to a Git repo, do something like the following, replacing {version tag} with the version of Dashboard you originally installed:

```
git init
rm .gitignore
wget https://raw.github.com/puppetlabs/puppet-dashboard/{version
tag}/.gitignore
git add .
git commit -m "conversion commit"
git branch original
git remote add upstream git://github.com/puppetlabs/puppet-dashboard.git
git fetch upstream
git reset --hard tags/{version tag}
git merge --no-ff original
git reset --soft tags/{version tag}
git stash save "Non-ignored files which were changed after the original
installation."
git checkout tags/v1.2.0
git stash apply
```

As with a standard Git upgrade, you'll need to re-chown all Dashboard files to the puppet-dashboard user:

```
$ sudo chown -R puppet-dashboard:puppet-dashboard ./*
```

Running Database Migrations

Puppet Dashboard's database schema changes as features are added and improved, and it needs to be updated after an upgrade. You may want to backup your database before you do this — see the database backups section of the maintaining chapter for further details.

DB migrations are done with a rake task, and should be simple and painless when upgrading between any two official releases of Dashboard.

```
$ sudo -u puppet-dashboard rake db:migrate RAILS_ENV=production
```

Remember that Rails does not consider "production" its default environment, so you must specify it manually for all rake tasks unless your `RAILS_ENV` environment variable is set or you are using the same database in the production and development environments.

You'll need to run `db:migrate` once for each environment you use. The `db:migrate` task can be safely run multiple times in the same environment.

After upgrading the code and the database, be sure to restart Dashboard's webserver and delayed jobs workers.

Upgrading From Versions Prior to 1.2.0

For reasons of speed and scalability, Dashboard 1.2 introduced a delayed job processing system. Dashboard won't lose any data sent by puppet masters if you don't run these delayed jobs, but they're necessary for analyzing reports and keeping the web UI up-to-date. You'll need to configure and run at least one worker process, and we recommend running exactly one process per CPU core.

Currently, the best way to manage these processes is with the `script/delayed_job` command, which can daemonize as a supervisor process and manage the requested number of workers. To start four workers and the monitor process:

```
$ sudo -u puppet-dashboard env RAILS_ENV=production script/delayed_job -p
dashboard -n 4 -m start
```

See the delayed jobs section of the installation chapter for more information.

Upgrading From Versions Prior to 1.1.0

In version 1.1.0, Dashboard changed the way it stores reports, and any reports from the 1.0.x series will have to be converted before they can be displayed or analyzed by the new version.

Since this can potentially take a long time, depending on your installation's report history, it isn't performed when running `rake db:migrate`. Instead, you should run:

```
$ sudo -u puppet-dashboard rake reports:schematize RAILS_ENV=production
```

This task will convert the most recent reports first, and if it is interrupted, it can be resumed by just re-running the command.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

Configuring Puppet Dashboard

This is a chapter of the Puppet Dashboard 1.2 manual.

NAVIGATION

Overview

Now that you've installed Dashboard and prepared it for basic production-level use, you can configure it to:

- Enable advanced features
- Increase security
- Improve performance
- Install plugins

Note that all rake tasks should be performed from a shell in the directory that contains Dashboard's code. Any relative paths mentioned below refer to locations within this directory. If you are running Dashboard in the recommended "production" environment, note that Rails does not consider production its default environment, and you must specify it manually with the `RAILS_ENV` environment variable when running any rake tasks.

Advanced Features

By default, Dashboard only responds to requests from a user or a puppet master. However, if you allow it to pull data from your puppet master, you can enable two extra features: the inventory service, and the file viewer.

Generating Certs and Connecting to the Puppet Master

Puppet uses SSL certificates to control who can make requests to the puppet master, so Dashboard has to obtain a signed cert before asking for facts or files. To do this, edit `config/settings.yml` to ensure that the `ca_server` and `ca_port` settings match the address and port of your puppet master, then run the following commands:

```
$ sudo -u puppet-dashboard rake cert:create_key_pair
$ sudo -u puppet-dashboard rake cert:request
```

You'll need to sign the certificate request on the master by running `puppet cert sign dashboard`. Then, from Dashboard's directory again, run:

```
$ sudo -u puppet-dashboard rake cert:retrieve
```

Enabling Inventory Support

With inventory support, Dashboard can display a complete list of facts on each node's detail page. It also adds a new "Inventory Search" page which can search your entire site for nodes matching a fact query.

Requirements: To use the inventory, you must be using Puppet 2.6.7 or later, configured to provide the inventory service. If you are running Puppet 2.7.12 or later, you have the option of using PuppetDB instead of the `inventory_active_record` backend.

Once the puppet master is properly configured with a database-backed inventory, edit your puppet master's `auth.conf` file to grant Dashboard find and search access to /facts:

```
path /facts
auth yes
method find, search
allow dashboard
```

Then, edit Dashboard's `config/settings.yml` to set `enable_inventory_service` to `true` and point `inventory_server` and `inventory_port` to your puppet master. Restart Dashboard, and node pages should now contain lists of facts.

Enabling the Filebucket Viewer

With the filebucket viewer, Dashboard can display the contents of different file versions when you click on MD5 checksums in reports.

Requirements: To use the filebucket viewer, you must be using Puppet 2.6.5 or later and your agent nodes must be configured to back up all files to a remote filebucket; this is done in your puppet master's `site.pp` manifest, where you must define a filebucket resource named "main"…

```
filebucket { "main":
  server => "{your puppet master}",
  path => false,
}
```

…and set a global resource default of…

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

If you are using inspect reports for a compliance workflow, you must also set `archive_files = true` in each agent's `puppet.conf`.

Once the site manifest has been properly configured, edit Dashboard's `config/settings.yml` to set `use_file_bucket_diffs` to `true` and point `file_bucket_server` and `file_bucket_port` to your puppet master. Restart Dashboard, and you should be able to view the contents of any file mentioned in a report by clicking on its MD5 checksum. Diffs are not currently enabled, but will appear in a future version of Dashboard.

Security

As Dashboard provides access to sensitive information and can make changes to your Puppet-managed infrastructure, you'll need some way to restrict access to it. Dashboard does not yet provide authentication or authorization, so you'll need to use external tools to secure it. Some options include:

- Host firewalling — The Dashboard server's firewall (e.g. `iptables`) can be used to limit which hosts can access the port Dashboard runs on.
- `stunnel` or `ssh` tunneling — You can use tunneling to provide an encrypted connection between hosts, e.g. if the Puppet Master and Puppet Dashboard are running on separate hosts. It can also allow you to access the web interface from a workstation once you've restricted access by IP.
- HTTP Basic Authentication — When serving Dashboard via Apache, you can require a username and password to access its URLs by setting authentication rules for `/` in Dashboard's vhost configuration:

  ```
  <Location "/">
    Order allow,deny
    Allow from 192.168.240.110 # your puppet master's IP
    Satisfy any
    AuthName "Puppet Dashboard"
    AuthType Basic
    AuthUserFile /etc/apache2/htpasswd
    Require valid-user
  </Location>
  ```

  Notice that you need to leave an access exception for your puppet master(s). Although it's possible to configure Puppet to use a password when connecting to Dashboard (by adding a username and password to Puppet's `reporturl` and the URL used by the `external_nodes` script), this currently requires patching Puppet's `http` report handler; see issue 7173 for more details.

- HTTPS (SSL) Encryption — When serving Dashboard via Apache, you can encrypt traffic between Puppet and the Dashboard. Using this requires a set of signed certificates from the puppet master — see generating certs and connecting to the puppet master for how to obtain them. The example configuration in `ext/passenger/dashboard-vhost.conf` includes a commented-out vhost configured to use SSL. You may need to change the Apache directives `SSLCertificateFile`, `SSLCertificateKeyFile`, `SSLCACertificateFile`, and `SSLCARevocationFile` to the paths of the files created by the `cert` rake tasks.

  If you have Dashboard set up to use HTTPS, you'll need to add an `https` prefix to the `DASHBOARD_URL` in the `external_node` script and potentially correct the port number (443, by default). You may also need to change the `CERT_PATH`, `PKEY_PATH`, and `CA_PATH` variables if your puppet master's hostname is not `puppet` or if your ssldir is not `/etc/puppet/ssl`.

  In order for reporting to work correctly via SSL, you will have to be running puppet master via Passenger or some other app server/webserver combination that can handle SSL; reporting to an SSL Dashboard is not supported when running puppet master under WEBrick. You'll also have to change the `reporturl` setting in `puppet.conf` to start with "https" instead of "http".

  This information may be outdated, and is currently being checked for accuracy.

Performance

Puppet Dashboard slows down as it manages more data. Here are ways to make it run faster, from easiest to hardest:

- Run exactly one `delayed_job` worker per CPU core.
- Make sure Dashboard is running in a production-quality web server, like Apache with Passenger.
- Make sure Dashboard is running in the production environment. Although Passenger runs Rails apps in production mode by default, other Rails tools may default to the much slower development environment.
- Optimize your database once a month; create a cron job that runs `rake RAILS_ENV=production db:raw:optimize` from your Puppet Dashboard directory. This will reorganize and reanalyze your database for faster queries.
- Tune the number of processes Dashboard uses to handle more concurrent requests. If you're using Apache with Phusion Passenger to serve Dashboard (as covered in the Installing chapter), you can modify the appropriate settings in Dashboard's vhost file; in particular, pay attention to the `PassengerHighPerformance`, `PassengerMaxPoolSize`, `PassengerPoolIdleTime`, `PassengerMaxRequests`, and `PassengerStatThrottleRate` settings.
- Regularly prune your old reports; see "cleaning old reports" in the maintenance chapter for more details.
- Run on a machine with a fast, local database.
- Run on a machine with enough processing power and memory.

- Run on a machine with fast backplane, controllers, and disks.

Installing Plugins

Puppet Labs plans to ship a variety of free and commercial plugins for Dashboard, which will add new features to support specific workflows. If you are installing a plugin, it probably came with official packages and its own installation instructions, but some general guidelines follow:

When installing a plugin from an official package, its files should be moved into the proper place with the proper ownership. However, you will probably have to run the `db:migrate` rake task after the installation is complete.

To install a plugin from source, rather than a package, you'll have to know the hardcoded internal name of the plugin. This should be listed in its documentation. Copy the plugin's directory to `vendor/plugins`, rename it to its proper internal name, and chown the directory and its files to the Dashboard user. Then, run the `puppet:plugin:install` task, passing the environment you're using[1] and the name of the plugin as variables:

```
$ sudo -u puppet-dashboard rake puppet:plugin:install PLUGIN=name
RAILS_ENV=production
```

After this, the plugin should be available and functioning. If you've been using Git to install and upgrade Dashboard, it should leave all plugin files untouched the next time you upgrade.

Uninstalling Plugins

This section will be filled in at a later date.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

1. `Puppet:plugin:install` runs `db:migrate` at the end. If you run in multiple environments regularly, you'll need to run `rake db:migrate` again for each additional one. ↩

Maintaining Puppet Dashboard

This is a chapter of the Puppet Dashboard 1.2 manual.

Overview

Puppet Dashboard exposes most of its functionality through its web UI, but it has a number of routine tasks that have to be performed on the command line by an admin. This chapter is a brief tour of some of these tasks.

Note that all rake tasks should be performed from a shell in the directory that contains Dashboard's code. Any relative paths mentioned below refer to locations within this directory. If you are running Dashboard in the recommended "production" environment, note that Rails does not consider production its default environment, and you must specify it manually with the `RAILS_ENV` environment variable when running any rake tasks.

Importing Pre-existing Reports

If your puppet master has stored a large number of reports from before your Dashboard came online, you can import them into Dashboard to get a better view into your site's history. If you are running Dashboard on the same server as your puppet master and its reports are stored in `/var/puppet/lib/reports`, you can simply run:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production reports:import
```

Alternately, you can copy the reports to your Dashboard server and run:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production reports:import
REPORT_DIR=/path/to/your/reports
```

Note that this task can take a very long time, depending on the number of reports to be imported. You can, however, safely interrupt and re-run the task, as the importer will automatically skip reports that Dashboard has already imported.

Optimizing the Database

Since Dashboard turns over a lot of data, its MySQL database should be periodically optimized for

speed and disk space. Dashboard has a rake task for doing this:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production db:raw:optimize
```

You should optimize Dashboard's database monthly, and we recommend creating a cron job to do so.

InnoDB is Taking Up Too Much Disk Space

Over time, the innodb database can get quite hefty, especially in larger deployments with many nodes. In some cases it can get large enough to consume all the space in `var`, which makes bad things happen. When this happens, you can follow the steps below to slim it back down.

1. Move your existing data to a backup file by running: `# mysqldump -p --databases {DASHBOARD'S DATABASE} {INVENTORY SERVICE DB (if present)} > /path/to/backup.sql`
2. Stop the MySQL service
3. Remove just the dashboard-specific database files. If you have no other databases, you can run `# cd /var/lib/mysql # rm -rf ./ib* # rm -rf ./console*`. Warning: this will remove everything, including any db's you may have added.
4. Restart the MySQL service.
5. Create new, empty databases by running this rake task: `# rake -f <FULL PATH TO DASHBOARD'S DIRECTORY>/Rakefile RAILS_ENV=production db:reset`.
6. Repopulate the databases by importing the data from the backup you created in step 1 by running: `# mysql -p < /path/to/backup.sql`.

Cleaning Old Reports

Reports will build up over time, which can slow Dashboard down. 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 older than 1 month:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production reports:prune upto=1
unit=mon
```

If you run 'rake reports:prune' without any arguments, it will display further usage instructions.

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.

A simple cron job to run the task monthly can be installed by running:

```
$ sudo -u puppet-dashboard rake cron:cleanup
```

Reading Logs

Dashboard may fail to start or display warnings if it is misconfigured or encounters an error. Details about these errors are recorded to log files that will help diagnose and resolve the problem.

You can find the logs in Dashboard's `log/` directory. You can customize your log rotation in `config/environment.rb` to devote more or less disk space to them.

If you're running Dashboard using Apache and Phusion Passenger, the Apache logs will contain higher-level information, such as severe errors that prevent Passenger from starting the application.

Database backups

Although you can back up and restore Dashboard's database with any tools, there are a pair of rake tasks which simplify the process.

Dumping the Database

To dump the Puppet Dashboard `production` database to a file called `production.sql`:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production db:raw:dump
```

Or dump it to a specific file:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production FILE=/my/backup/file.sql
db:raw:dump
```

Restoring the Database

To restore the Puppet Dashboard from a file called `production.sql` to your `production` environment:

```
$ sudo -u puppet-dashboard rake RAILS_ENV=production FILE=production.sql
db:raw:restore
```

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard

- Maintaining Dashboard
- Using Dashboard
- Rake API

Using Puppet Dashboard

This is a chapter of the Puppet Dashboard 1.2 manual.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

This chapter has not been written. However, similar documentation has been written for the Puppet Enterprise 2.0 console. The PE console is built on Puppet Dashboard and adds functionality, but the features they have in common are largely identical save for branding and cosmetic changes.

The three chapters below document the features that Dashboard and the PE console have in common:

- Navigating the Console
- Viewing Reports and Inventory Data
- Grouping and Classifying Nodes

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

Puppet Dashboard's Rake API

This is a chapter of the Puppet Dashboard 1.2 manual.

NAVIGATION

- Installing Dashboard
- Upgrading Dashboard
- Configuring Dashboard
- Maintaining Dashboard
- Using Dashboard
- Rake API

Rake API

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

This list of tasks applies to Dashboard 1.2.4 and later. Some of these tasks were not included in prior releases of Dashboard 1.2.

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

```
$ sudo -u puppet-dashboard rake -f <FULL PATH TO DASHBOARD'S
DIRECTORY>/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.
