PuppetDB 1 User's Guide

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

PuppetDB 1 » Overview

PuppetDB collects data generated by Puppet. It enables advanced Puppet features like the inventory service and exported resources, and can be the foundation for other applications that use Puppet's data.

Install It Now

To start using PuppetDB today:

- Review the system requirements below (and, optionally, our scaling recommendations).
- Follow the installation instructions: easy install on our recommended platforms or advanced install on any other *nix.
- Connect your puppet master to PuppetDB (or at standalone puppet apply sites, connect every node to PuppetDB).

Version Note

This manual covers the 1.x series of PuppetDB releases, which has several changes from the 0.9 beta release series. See the release notes for information on all changes since the final 0.9.x release.

Note that, due to issues with package repositories during the run-up to the 1.0 release, the following non-1.x version numbers are actually considered 1.0 release candidates:

- 0.11.0 (actually 1.0rc2)
- 0.10.0 (actually 1.0rc1)

What Data?

In version 1, PuppetDB stores:

- The most recent facts from every node
- The most recent catalog for every node

Together, these give you a huge inventory of metadata about every node in your infrastructure and a searchable database of every single resource being managed on any node.

Puppet itself can search a subset of this data using exported resources, which allow nodes to manage resources on other nodes. This is similar to the capabilities of the legacy ActiveRecord `storeconfigs` interface, but much, much faster. The remaining data is available through PuppetDB's query APIs (see navigation sidebar for details) and Puppet's inventory service.

What Next?

In future versions, PuppetDB will store reports and historical catalogs for every node.

System Requirements

*nix Server with JDK 1.6+

STANDARD INSTALL: RHEL, CENTOS, DEBIAN, UBUNTU, OR FEDORA

Puppet Labs provides packages for PuppetDB which simplify setup of its SSL certificates and init scripts. These packages are available for the following operating systems:

- Red Hat Enterprise Linux 5 or 6 or any distro derived from it (including CentOS)
- Debian Squeeze, Lenny, Wheezy, or Sid
- Ubuntu 12.04 LTS, 10.04 LTS, 8.04 LTS, 11.10, or 11.04
- Fedora 15 or 16

See here for installation instructions.

CUSTOM INSTALL: ANY UNIX-LIKE OS

If you're willing to do some manual configuration, PuppetDB can run on any Unix-like OS with JDK 1.6 or higher, including:

- Recent MacOS X versions (using built-in Java support)
- Nearly any Linux distribution (using vendor's OpenJDK packages)
- Nearly any *nix system running a recent Oracle-provided JDK

See here for advanced installation instructions.

Puppet 2.7.12

Your site's puppet masters must be running Puppet 2.7.12 or later. You will need to connect your puppet masters to PuppetDB after installing it. If you wish to use PuppetDB with standalone nodes that are running puppet apply, every node must be running 2.7.12 or later.

Robust Hardware

PuppetDB will be a critical component of your Puppet deployment and so should be run on a robust and reliable server.

However, it can do a lot with fairly modest hardware: in benchmarks using real-world catalogs from a customer, a single 2012 laptop (16 GB of RAM, consumer-grade SSD, and quad-core processor) running PuppetDB and PostgreSQL was able to keep up with sustained input from 8,000 simulated Puppet nodes checking in every 30 minutes. Powerful server-grade hardware will perform even better.

The actual requirements will vary wildly depending on your site's size and characteristics. At smallish sites, you may even be able to run PuppetDB on your puppet master server.

For more on fitting PuppetDB to your site, see "Scaling Recommendations."

Open Source

PuppetDB is developed openly, and is released under the Apache 2.0 license. You can get the source — and contribute to it! — at the PuppetDB GitHub repo. Bugs and feature requests are welcome at Puppet Labs's issue tracker.

PuppetDB 1 » Release Notes

The following notable changes have been made since version 0.9.2 of PuppetDB:

1.0.2

Many thanks to the following people who contributed patches to this release:

- Matthaus Owens

Fixes:

- (#17178) Update rubylib on debian/ubuntu installs

  Previously the terminus would be installed to the 1.8 sitelibdir for ruby1.8 or the 1.9.1 vendorlibdir on ruby1.9. The ruby1.9 code path was never used, so platforms with ruby1.9 as the default (such as quantal and wheezy) would not be able to load the terminus. Modern debian packages put version agnostic ruby code in vendordir (/usr/lib/ruby/vendor_ruby), so this commit moves the terminus install dir to be vendordir.

1.0.1

Many thanks to the following people who contributed patches to this release:

- Deepak Giridharagopal
- Nick Lewis
- Matthaus Litteken
- Chris Price

Fixes:

- (#16180) Properly handle edges between exported resources

  This was previously failing when an edge referred to an exported resource which was also collected, because it was incorrectly assuming collected resources would always be marked as NOT exported. However, in the case of a node collecting a resource which it also exports, the resource is still marked exported. In that case, it can be distinguished from a purely exported resource by whether it's virtual. Purely virtual, non-exported resources never appear in the catalog.

  Virtual, exported resources are not collected, whereas non-virtual, exported resources are. The former will eventually be removed from the catalog before being sent to the agent, and thus aren't eligible for participation in a relationship. We now check whether the resource is virtual rather than exported, for correct behavior.

- (#16535) Properly find edges that point at an exec by an alias

  During namevar aliasing, we end up changing the :alias parameter to 'alias' and using that for the duration (to distinguish "our" aliases form the "original" aliases). However, in the case of exec, we were bailing out early because execs aren't isomorphic, and not adding 'alias'. Now we will always change :alias to 'alias', and just won't add the namevar alias for execs.

- (#16407) Handle trailing slashes when creating edges for file resources

  We were failing to create relationships (edges) to File resources if the relationship was specified with a different number of trailing slashes in the title than the title of the original resource.

- (#16652) Replace dir with specific files for terminus package

  Previously, the files section claimed ownership of Puppet's libdir, which confuses rpm when both

packages are installed. This commit breaks out all of the files and only owns one directory, which clearly belongs to puppetdb. This will allow rpm to correctly identify files which belong to puppet vs puppetdb-terminus.

1.0.0

The 1.0.0 release contains no changes from 0.11.0 except a minor packaging change.

0.11.0

Many thanks to the following people who contributed patches to this release:

- Kushal Pisavadia
- Deepak Giridharagopal
- Nick Lewis
- Moses Mendoza
- Chris Price

Notable features:

- Additional database indexes for improved performance

  Queries involving resources (type,title) or tags without much additional filtering criteria are now much faster. Note that tag queries cannot be sped up on PostgreSQL 8.1, as it doesn't have support for GIN indexes on array columns.

- Automatic generation of heap snapshots on OutOfMemoryError

  In the unfortunate situation where PuppetDB runs out of memory, a heap snapshot is automatically generated and saved in the log directory. This helps us work with users to much more precisely triangulate what's taking up the majority of the available heap without having to work to reproduce the problem on a completely different system (an often difficult proposition). This helps keep PuppetDB lean for everyone.

- Preliminary packaging support for Fedora 17 and Ruby 1.9

  This hasn't been fully tested, nor integrated into our CI systems, and therefore should be considered experimental. This fix adds support for packaging for ruby 1.9 by modifying the @plibdir path based on the ruby version. `RUBY_VER` can be passed in as an environment variable, and if none is passed, `RUBY_VER` defaults to the ruby on the local host as reported by facter. As is currently the case, we use the sitelibdir in ruby 1.8, and with this commit use vendorlibdir for 1.9. Fedora 17 ships with 1.9, so we use this to test for 1.9 in the spec file. Fedora 17 also ships with open-jdk 1.7, so this commit updates the Requires to 1.7 for fedora 17.

- Resource tags semantics now match those of Puppet proper

  In Puppet, tags are lower-case only. We now fail incoming catalogs that contain mixed case tags, and we treat tags in queries as case-insensitive comparisons.

Notable fixes:

- Properly escape resource query strings in our terminus

  This fixes failures caused by storeconfigs queries that involve, for example, resource titles whose names contain spaces.

- (#15947) Allow comments in puppetdb.conf

  We now support whole-line comments in puppetdb.conf.

- (#15903) Detect invalid UTF-8 multi-byte sequences

  Prior to this fix, certain sequences of bytes used on certain versions of Puppet with certain versions of Ruby would cause our terminii to send malformed data to PuppetDB (which the daemon then properly rejects with a checksum error, so no data corruption would have taken place).

- Don't remove puppetdb user during RPM package uninstall

  We never did this on Debian systems, and most other packages seem not to as well. Also, removing the user and not all files owned by it can cause problems if another service later usurps the user id.

- Compatibility with legacy storeconfigs behavior for duplicate resources

  Prior to this commit, the puppetdb resource terminus was not setting a value for "collector_id" on collected resources. This field is used by puppet to detect duplicate resources (exported by multiple nodes) and will cause a run to fail. Hence, the semantics around duplicate resources were ill-specified and could cause problems. This fix adds code to set the collector id based on node name + resource title + resource type, and adds tests to verify that a puppet run will fail if it collects duplicate instances of the same resource from different exporters.

- Internal benchmarking suite fully functional again

  Previous changes had broken the benchmark tool; functionality has been restored.

- Better version display

  We now display the latest version info during daemon startup and on the web dashboard.

0.10.0

Many thanks to the following people who contributed patches to this release:

- Deepak Giridharagopal
- Nick Lewis
- Matthaus Litteken
- Moses Mendoza
- Chris Price

Notable features:

- Auto-deactivation of stale nodes

  There is a new, optional setting you can add to the `[database]` section of your configuration: `node-ttl-days`, which defines how long, in days, a node can continue without seeing new activity (new catalogs, new facts, etc) before it's automatically deactivated during a garbage-collection run.

  The default behavior, if that config setting is ommitted, is the same as in previous releases: no automatic deactivation of anything.

  This feature is useful for those who have a non-trivial amount of volatility in the lifecycles of

their nodes, such as those who regularly bring up nodes in a cloud environment and tear them down shortly thereafter.

- (#15696) Limit the number of results returned from a resource query

  For sites with tens or even hundreds of thousands of resources, an errant query could result in PuppetDB attempting to pull in a large number of resources and parameters into memory before serializing them over the wire. This can potentially trigger out-of-memory conditions.

  There is a new, optional setting you can add to the `[database]` section of your configuration: `resource-query-limit`, which denotes the maximum number of resources returnable via a resource query. If the supplied query results in more than the indicated number of resources, we return an HTTP 500.

  The default behavior is to limit resource queries to 20,000 resources.

- (#15696) Slow query logging

  There is a new, optional setting you can add to the `[database]` section of your configuration: `log-slow-statements`, which denotes how many seconds a database query can take before the query is logged at WARN level.

  The default behavior for this setting is to log queries that take more than 10 seconds.

- Add support for a –debug flag, and a debug-oriented startup script

  This commit adds support for a new command-line flag: –debug. For now, this flag only affects logging: it forces a console logger and ensures that the log level is set to DEBUG. The option is also added to the main config hash so that it can potentially be used for other purposes in the future.

  This commit also adds a shell script, `puppetdb-foreground`, which can be used to launch the services from the command line. This script will be packaged (in /usr/sbin) along with the puppetdb-ssl-setup script, and may be useful in helping users troubleshoot problems on their systems (especially problems with daemon startup).

Notable fixes:

- Update CONTRIBUTING.md to better reflect reality

  The process previously described in CONTRIBUTING.md was largely vestigial; we've now updated that documentation to reflect the actual, current contribution process.

- Proper handling of composite namevars

  Normally, as part of converting a catalog to the PuppetDB wire format, we ensure that every resource has its namevar as one of its aliases. This allows us to handle edges that refer to said resource using its namevar instead of its title.

  However, Puppet implements `#namevar` for resources with composite namevars in a strange way, only returning part of the composite name. This can result in bugs in the generated catalog, where we may have 2 resources with the same alias (because `#namevar` returns the same thing for both of them).

  Because resources with composite namevars can't be referred to by anything other than their title when declaring relationships, there's no real point to adding their aliases in anyways. So

now we don't bother.

- Fix deb packaging so that the puppetdb service is restarted during upgrades

  Prior to this commit, when you ran a debian package upgrade, the puppetdb service would be stopped but would not be restarted.

- (#1406) Add curl-based query examples to docs

  The repo now contains examples of querying PuppetDB via curl over both HTTP and HTTPS.

- Documentation on how to configure PuppetDB to work with "puppet apply"

  There are some extra steps necessary to get PuppetDB working properly with Puppet apply, and there are limitations thereafter. The repo now contains documentation around what those limitations are, and what additional configuration is necessary.

- Upgraded testing during acceptance test runs

  We now automatically test upgrades from the last published version of PuppetDB to the currently-under-test version.

- (#15281) Added postgres support to acceptance testing

  Our acceptance tests now regularly run against both the embedded database and PostgreSQL, automatically, on every commit.

- (#15378) Improved behavior of acceptance tests in single-node environment

  We have some acceptance tests that require multiple nodes in order to execute successfully (mostly around exporting / collecting resources). If you tried to run them in a single-node environment, they would give a weird ruby error about 'nil' not defining a certain method. Now, they will be skipped if you are running without more than one host in your acceptance test-bed.

- Spec tests now work against Puppet master branch

  We now regularly and automatically run PuppetDB spec tests against Puppet's master branch.

- Acceptance testing for RPM-based systems

  Previously we were running all of our acceptance tests solely against Debian systems. We now run them all, automatically upon each commit against RedHat machines as well.

- Added new `rake version` task

  Does what it says on the tin.

PuppetDB » Known Issues

Bugs and Feature Requests

PuppetDB's bugs and feature requests are managed in Puppet Labs's issue tracker. Search this database if you're having problems and please report any new issues to us!

Broader Issues

Autorequire relationships are opaque

Puppet resource types can "autorequire" other resources when certain conditions are met but we

don't correctly model these relationships in PuppetDB. (For example, if you manage two file resources where one is a parent directory of the other, Puppet will automatically make the child dependent on the parent.) The problem is that these dependencies are not written to the catalog; puppet agent creates these relationships on the fly when it reads the catalog. Getting these relationships into PuppetDB will require a significant change to Puppet's core.

PuppetDB 1 » Community Projects and Add-ons

None of the following projects are published by or endorsed by Puppet Labs. They are linked to here for informational purposes only.

Jason Hancock — nagios-puppetdb

A collection of Nagios scripts/plugins for monitoring PuppetDB. These plugins get data using PuppetDB's metrics APIs. Pulling this data into Nagios lets you monitor key metrics over time and receive alerts when they cross certain thresholds. This can partially or completely replace the built-in performance dashboard. Especially useful for knowing when the heap size or thread count needs tuning.

Eric Dalén — PuppetDB query functions for Puppet

A Puppet module with functions for querying PuppetDB data. By default, exported resources are the only way for Puppet manifests to get other nodes' data from PuppetDB. These functions let you get other data. In particular, the `pdbnodequery` function can let you search nodes by class or resource, an operation that normally requires multiple PuppetDB queries. The functions in this module include:

- `pdbresourcequery`
- `pdbnodequery`
- `pdbfactquery`
- `pdbstatusquery`
- `pdbquery`

PuppetDB 1 » Installing PuppetDB

Notes:

- After following these instructions, you should connect your puppet master(s) to PuppetDB. (If you use a standalone Puppet deployment, you will need to connect every node to PuppetDB.)
- These instructions are for platforms with official PuppetDB packages. To install on other systems, you should instead follow the instructions for installing from source.
- If this is a production deployment, review the scaling recommendations before installing. You should ensure your PuppetDB server will be able to comfortably handle your site's load.

Step 1: Install and Configure Puppet

If Puppet isn't fully installed and configured yet on your PuppetDB server, install it and request/sign/retrieve a certificate for the node:

- Instructions for Puppet Enterprise
- Instructions for open source Puppet

Your PuppetDB server should be running puppet agent and have a signed certificate from your puppet master server. If you run `puppet agent --test`, it should successfully complete a run, ending with "`notice: Finished catalog run in X.XX seconds`."

> Note: If Puppet doesn't have a valid certificate when PuppetDB is installed, you will have to run the SSL config script and edit the config file, or manually configure PuppetDB's SSL credentials before the puppet master will be able to connect to PuppetDB.

Step 2: Enable the Puppet Labs Package Repository

If you didn't already use it to install Puppet, you will need to enable the Puppet Labs package repository for your system. Follow the instructions linked below, then continue with step 3 of this guide:

- Instructions for PE users
- Instructions for open source users

Step 3: Install PuppetDB

Use Puppet to install PuppetDB.

For PE Users

```
$ sudo puppet resource package pe-puppetdb ensure=latest
```

For Open Source Users

```
$ sudo puppet resource package puppetdb ensure=latest
```

Step 4: Configure Database

If this is a production deployment, you should confirm and configure your database settings:

- Deployments of 100 nodes or fewer can continue to use the default built-in database backend, but should increase PuppetDB's maximum heap size to at least 1 GB.
- Large deployments over 100 nodes should set up a PostgreSQL server and configure PuppetDB to use it. You may also need to adjust the maximum heap size.

You can change PuppetDB's database at any time, but note that changing the database does not migrate PuppetDB's data, so the new database will be empty. However, as this data is automatically

generated many times a day, PuppetDB should recover in a relatively short period of time.

Step 5: Start the PuppetDB Service

Use Puppet to start the PuppetDB service and enable it on startup.

For PE Users

```
$ sudo puppet resource service pe-puppetdb ensure=running enable=true
```

For Open Source Users

```
$ sudo puppet resource service puppetdb ensure=running enable=true
```

You must also configure your PuppetDB server's firewall to accept incoming connections on port 8081.

> PuppetDB is now fully functional and ready to receive catalogs and facts from any number of puppet master servers.

Finish: Connect Puppet to PuppetDB

You should now configure your puppet master(s) to connect to PuppetDB.

If you use a standalone Puppet site, you should configure every node to connect to PuppetDB.

Troubleshooting Installation Problems

- Check the log file, and see whether PuppetDB knows what the problem is. This file will be either `/var/log/puppetdb/puppetdb.log` or `/var/log/pe-puppetdb/pe-puppetdb.log`.
- If PuppetDB is running but the puppet master can't reach it, check PuppetDB's jetty configuration to see which port(s) it is listening on, then attempt to reach it by telnet (`telnet <host> <port>`) from the puppet master server. If you can't connect, the firewall may be blocking connections. If you can, Puppet may be attempting to use the wrong port, or PuppetDB's keystore may be misconfigured (see below).
- Check whether any other service is using PuppetDB's port and interfering with traffic.
- Check PuppetDB's jetty configuration and the `/etc/puppetdb/ssl` (or `/etc/pe-puppetdb/ssl`) directory, and make sure it has a truststore and keystore configured. If it didn't create these during installation, you will need to run the SSL config script and edit the config file or manually configure a truststore and keystore before a puppet master can contact PuppetDB.

PuppetDB 1 » Connecting Puppet Masters to PuppetDB

> Note: To use PuppetDB, your site's puppet master(s) must be running Puppet 2.7.12 or later .

After PuppetDB is installed and running, you should configure your puppet master(s) to use it. Once

connected to PuppetDB, puppet masters will do the following:

- Send every node's catalog to PuppetDB
- Send every node's facts to PuppetDB
- Query PuppetDB when compiling node catalogs that collect exported resources
- Query PuppetDB when responding to inventory service requests

Working on your puppet master server(s), follow all of the instructions below:

Step 1: Install Plugins

Currently, puppet masters need additional Ruby plugins in order to use PuppetDB. Unlike custom facts or functions, these cannot be loaded from a module and must be installed in Puppet's main source directory.

For PE Users

Enable the Puppet Labs repo and then install the `pe-puppetdb-terminus` package:

```
$ sudo puppet resource package pe-puppetdb-terminus ensure=latest
```

For Open Source Users

Enable the Puppet Labs repo and then install the `puppetdb-terminus` package:

```
$ sudo puppet resource package puppetdb-terminus ensure=latest
```

On Platforms Without Packages

If your puppet master isn't running Puppet from a supported package, you will need to install the plugins manually:

- Download the PuppetDB source code, unzip it and navigate into the resulting directory in your terminal.
- Run `sudo cp -R puppet/lib/puppet /usr/lib/ruby/site_ruby/1.8/puppet`. Replace the second path with the path to your Puppet installation if you have installed it somewhere other than `/usr/lib/ruby/site_ruby`.

Step 2: Edit Config Files

Locate Puppet's Config Directory

Find your puppet master's config directory by running `sudo puppet config print confdir`. It will usually be at either `/etc/puppet/` or `/etc/puppetlabs/puppet/`.

You will need to edit (or create) three files in this directory:

1. Edit puppetdb.conf

The puppetdb.conf file will probably not exist yet. Create it, and add the PuppetDB server's hostname and port:

```
[main]
```

```
server = puppetdb.example.com
port = 8081
```

- PuppetDB's port for secure traffic defaults to 8081.□
- Puppet requires use of PuppetDB's secure HTTPS port. You cannot use the unencrypted, plain HTTP port.

If no puppetdb.conf file exists, the following default values will be used:□

```
server = puppetdb
port = 8081
```

2. Edit puppet.conf

To enable PuppetDB for the inventory service and saved catalogs/exported resources, add the following settings to the `[master]` block of puppet.conf (or edit them if already present):

```
[master]
  storeconfigs = true
  storeconfigs_backend = puppetdb
```

Note: The `thin_storeconfigs` and `async_storeconfigs` settings should be absent or set to `false`. If you have previously used the puppet queue daemon (puppetqd), you should now disable it.

3. Edit routes.yaml

The routes.yaml file will probably not exist yet. Create it if necessary, and add the following:□

```
---
master:
  facts:
    terminus: puppetdb
    cache: yaml
```

This will make PuppetDB the authoritative source for the inventory service.

Step 3: Restart Puppet Master

Use your system's service tools to restart the puppet master service. For open source users, the command to do this will vary depending on the front-end web server being used. For Puppet Enterprise users, run:

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

Your puppet master should now be using PuppetDB to store and retrieve catalogs, facts, and exported resources. You can test this by triggering a puppet agent run on an arbitrary node, then logging into your PuppetDB server and viewing the `/var/log/puppetdb/puppetdb.log` or `/var/log/pe-puppetdb/pe-puppetdb.log` file — you should see calls to the "replace□

facts" and "replace catalog" commands:

```
2012-05-17 13:08:41,664 INFO  [command-proc-67] [puppetdb.command]
[85beb105-5f4a-4257-a5ed-cdf0d07aa1a5] [replace facts]
screech.example.com
2012-05-17 13:08:45,993 INFO  [command-proc-67] [puppetdb.command]
[3a910863-6b33-4717-95d2-39edf92c8610] [replace catalog]
screech.example.com
```

PuppetDB 1 » Connecting Standalone Puppet Nodes to PuppetDB

Note: To use PuppetDB, the nodes at your site must be running Puppet 2.7.12 or later.

PuppetDB can also be used with standalone Puppet deployments where each node runs `puppet apply`. Once connected to PuppetDB, `puppet apply` will do the following:

- Send the node's catalog to PuppetDB
- Query PuppetDB when compiling catalogs that collect exported resources

Note that standalone deployments can only store catalogs and cannot use the inventory service. This is due to a limitation in Puppet.

You will need to take the following steps to configure your standalone nodes to connect to PuppetDB. Note that since you must change Puppet's configuration on every managed node, we strongly recommend that you do so with Puppet itself.

Step 1: Configure SSL

PuppetDB requires client authentication for its SSL connections and the PuppetDB terminus plugins require SSL to talk to PuppetDB. You must configure Puppet and PuppetDB to work around this double-bind by using one of the following options:

Option A: Set Up an SSL Proxy for PuppetDB

1. Edit the `jetty` section of the PuppetDB config files to remove all SSL-related settings.
2. Install a general purpose web server (like Apache or Nginx) on the PuppetDB server.
3. Configure the web server to listen on port 8081 with SSL enabled and proxy all traffic to `localhost:8080` (or whatever unencrypted hostname and port were set in jetty.ini). The proxy server can use any certificate — as long as Puppet has never downloaded a CA cert from a puppet master, it will not verify the proxy server's certificate. If your nodes have downloaded CA certs, you must either make sure the proxy server's cert was signed by the same CA, or delete the CA cert.

More detailed instructions for setting up this proxy will be added to this guide at a later date.

Option B: Issue Certificates to All Puppet Nodes

When talking to PuppetDB, puppet apply can use the certificates issued by a puppet master's certificate authority . You can issue certificates to every node by setting up a puppet master server

with dummy manifests, running `puppet agent --test` once on every node, signing every certificate request on the puppet master, and running `puppet agent --test` again on every node.

Do the same on your PuppetDB node, then re-run the SSL setup script. PuppetDB will now trust connections from your Puppet nodes.

You will have to sign a certificate for every new node you add to your site.

Step 2: Install Terminus Plugins on Every Puppet Node

Currently, Puppet needs extra Ruby plugins in order to use PuppetDB. Unlike custom facts or functions, these cannot be loaded from a module and must be installed in Puppet's main source directory.

- First, ensure that the appropriate Puppet Labs package repository (Puppet Enterprise, or open source) is enabled. You can use a package resource to do this or use the apt::source (from the puppetlabs-apt module) and yumrepo types.
- Next, use Puppet to ensure that the `pe-puppetdb-terminus` or `puppetdb-terminus` package is installed:

```
# for PE:
package {'pe-puppetdb-terminus':
  ensure => installed,
}

# for open source:
package {'puppetdb-terminus':
  ensure => installed,
}
```

On Platforms Without Packages

If your puppet master isn't running Puppet from a supported package, you will need to install the plugins using file resources.

- Download the PuppetDB source code; unzip it, locate the `puppet/lib/puppet` directory and put it in the `files` directory of the Puppet module you are using to enable PuppetDB integration.
- Identify the install location of Puppet on your nodes.
- Create a file resource in your manifests for each of the plugin files, to move them into place on each node.

```
# <modulepath>/puppetdb/manifests/terminus.pp
class puppetdb::terminus {
  $puppetdir = "$rubysitedir/puppet"

  file {$puppetdir:
    ensure => directory,
    recurse => remote, # Copy these files without deleting the existing
files
    source => "puppet:///modules/puppetdb/puppet",
    owner => root,
    group => root,
    mode => 0644,
  }
}
```

Step 3: Manage Config Files on Every Puppet Node

All of the config files you need to manage will be in Puppet's config directory (`confdir`). When managing these files with puppet apply, you can use the `$settings::confdir` variable to automatically discover the location of this directory.

Manage puppetdb.conf

You can specify the contents of puppetdb.conf directly in your manifests. It should contain the PuppetDB server's hostname and port:

```
[main]
server = puppetdb.example.com
port = 8081
```

- PuppetDB's port for secure traffic defaults to 8081.
- Puppet requires use of PuppetDB's secure, HTTPS port. You cannot use the unencrypted, plain HTTP port.

If no puppetdb.conf file exists, the following default values will be used:

```
server = puppetdb
port = 8081
```

Manage puppet.conf

You will need to create a template for puppet.conf based on your existing configuration. Then, modify the template by adding the following settings to the `[main]` block:

```
[main]
  storeconfigs = true
  storeconfigs_backend = puppetdb
```

> Note: The `thin_storeconfigs` and `async_storeconfigs` settings should be absent or set to `false`.

Manage routes.yaml

Typically, you can specify the contents of routes.yaml directly in your manifests; if you are already using it for some other purpose, you will need to manage it with a template based on your existing configuration. Ensure that the following keys are present:

```
---
apply:
  facts:
    terminus: facter
    cache: facter
```

This will disable fact storage and prevent puppet apply from using stale facts.

PuppetDB 1 » Upgrading PuppetDB

Checking for Updates

PuppetDB's performance dashboard displays the current version in the upper right corner. It also automatically checks for updates and will show a link to the newest version under the version indicator if your deployment is out of date.

What to Upgrade

When a new version of PuppetDB is released, you will need to upgrade:

1. PuppetDB itself
2. The terminus plugins on every puppet master (or every node, if using a standalone deployment)

You should upgrade PuppetDB first. Since PuppetDB will be down for a few minutes during the upgrade and puppet masters will not be able to serve catalogs until it comes back, you should schedule upgrades during a maintenance window during which no new nodes will be brought on line.

If you upgrade PuppetDB without upgrading the terminus plugins, your Puppet deployment should continue to function identically, with no loss of functionality. However, you may not be able to take advantage of new PuppetDB features until you upgrade the terminus plugins.

Upgrading PuppetDB

On your PuppetDB server: stop the PuppetDB service, upgrade the PuppetDB package, then restart the PuppetDB service.

For PE Users

```
$ sudo puppet resource service pe-puppetdb ensure=stopped
$ sudo puppet resource package pe-puppetdb ensure=latest
$ sudo puppet resource service pe-puppetdb ensure=running
```

For Open Source Users

```
$ sudo puppet resource service puppetdb ensure=stopped
$ sudo puppet resource package puppetdb ensure=latest
$ sudo puppet resource service puppetdb ensure=running
```

On Platforms Without Packages

If you installed PuppetDB by running `rake install`, you should obtain a fresh copy of the source, stop the service, and run `rake install` again. Note that this workflow is not well tested; if you run into problems, please report them on the PuppetDB issue tracker.

If you are running PuppetDB from source, you should stop the service, replace the source, and start the service as described in the advanced installation guide.

Upgrading the Terminus Plugins

On your puppet master servers: upgrade the PuppetDB terminus plugins package, then restart the puppet master's web server:

For PE Users

```
$ sudo puppet resource package pe-puppetdb-terminus ensure=latest
$ sudo puppet resource service pe-httpd ensure=stopped
$ sudo puppet resource service pe-httpd ensure=running
```

For Open Source Users

```
$ sudo puppet resource package puppetdb-terminus ensure=latest
```

The command to restart the puppet master will vary depending on which web server you are using.

On Platforms Without Packages

Obtain a fresh copy of the PuppetDB source, and follow the instructions for installing the terminus plugins.

The command to restart the puppet master will vary depending on which web server you are using.

PuppetDB 1 » Installing PuppetDB from Source

If possible, we recommend installing PuppetDB from packages. However, if you are installing PuppetDB on a system not supported with official packages, or if you are testing or developing a new version of PuppetDB, you will need to install it from source.

Step 1: Install Prerequisites

Use your system's package tools to ensure that the following prerequisites are installed:

- Facter, version 1.6.8 or higher
- JDK 1.6 or higher
- Leiningen
- Git (for checking out the source code)

Step 2, Option A: Install from Source

Run the following commands:

```
$ mkdir -p ~/git && cd ~/git
$ git clone git://github.com/puppetlabs/puppetdb
$ cd puppetdb
$ sudo rake install
```

This will install PuppetDB, put a `puppetdb` init script in `/etc/init.d` and create a default configuration directory in `/etc/puppetdb`.

Step 2, Option B: Run Directly from Source

While installing from source is useful for simply running a development version for testing, for development it's better to be able to run directly from source, without any installation step.

Run the following commands:

```
$ mkdir -p ~/git && cd ~/git
$ git clone git://github.com/puppetlabs/puppetdb
$ cd puppetdb

# Download the dependencies
$ lein deps
```

This will let you develop on PuppetDB and see your changes by simply editing the code and restarting the server. It will not create an init script or default configuration directory. To start the PuppetDB service when running from source, you will need to run the following:

```
$ lein run services -c /path/to/config.ini
```

A sample config file is provided in the root of the source repo: `config.sample.ini`. You can also provide a conf.d-style directory instead of a flat config file.

Other useful commands for developers:

- `lein test` to run the test suite
- `lein docs` to build docs in `docs/uberdoc.html`

Step 3, Option A: Run the SSL Configuration Script

If your PuppetDB server has puppet agent installed, has received a valid certificate from your site's Puppet CA, and you installed PuppetDB from source, then PuppetDB can re-use Puppet's certificate.

Run the following command:

```
$ sudo /usr/sbin/puppetdb-ssl-setup
```

This will create a keystore and truststore in `/etc/puppetdb/ssl` and will print the password to both files in `/etc/puppetdb/ssl/puppetdb_keystore_pw.txt`.

You should now configure HTTPS in PuppetDB's config file(s); see below.

Step 3, Option B: Manually Create a Keystore and Truststore

If you will not be using Puppet on your PuppetDB server, you must manually create a certificate, a keystore, and a truststore. This is an involved process, so we highly recommend installing Puppet and using Option A above, even if you will not be using puppet agent to manage the PuppetDB server.

On the CA Puppet Master: Create a Certificate

Use `puppet cert generate` to create a certificate and private key for your PuppetDB server. Run the following, using your PuppetDB server's hostname:

```
$ sudo puppet cert generate puppetdb.example.com
```

Copy the Certificate to the PuppetDB Server

Copy the CA certificate, the PuppetDB certificate, and the PuppetDB private key to your PuppetDB server. Run the following on your CA puppet master server, using your PuppetDB server's hostname:

```
$ sudo scp $(puppet master --configprint ssldir)/ca/ca_crt.pem
puppetdb.example.com:/tmp/certs/ca_crt.pem
$ sudo scp $(puppet master --configprint
ssldir)/private_keys/puppetdb.example.com.pem
puppetdb.example.com:/tmp/certs/privkey.pem
$ sudo scp $(puppet master --configprint ssldir)/certs/puppetdb.example.com.pem
puppetdb.example.com:/tmp/certs/pubkey.pem
```

You may now log out of your puppet master server.

On the PuppetDB Server: Create a Truststore

On your PuppetDB server, navigate to the directory where you copied the certificates and keys:

```
$ cd /tmp/certs
```

Now use `keytool` to create a truststore file. A truststore contains the set of CA certs to use for validation.

```
# keytool -import -alias "My CA" -file ca_crt.pem -keystore truststore.jks
Enter keystore password:
Re-enter new password:
.
.
.
Trust this certificate? [no]:  y
Certificate was added to keystore
```

Note that you must supply a password. Remember the password you used, as you'll need it to configure PuppetDB later. Once imported, you can view your certificate:

```
# keytool -list -keystore truststore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

my ca, Mar 30, 2012, trustedCertEntry,
Certificate fingerprint (MD5): 99:D3:28:6B:37:13:7A:A2:B8:73:75:4A:31:78:0B:68
```

Note the MD5 fingerprint; you can use it to verify this is the correct cert:

```
# openssl x509 -in ca_crt.pem -fingerprint -md5
MD5 Fingerprint=99:D3:28:6B:37:13:7A:A2:B8:73:75:4A:31:78:0B:68
```

On the PuppetDB Server: Create a Keystore

In the same directory as the truststore you just created, use `keytool` to create a Java keystore. A keystore file contains certificates to use during HTTPS.

```
# cat privkey.pem pubkey.pem > temp.pem
# openssl pkcs12 -export -in temp.pem -out puppetdb.p12 -name
puppetdb.example.com
Enter Export Password:
Verifying - Enter Export Password:
# keytool -importkeystore  -destkeystore keystore.jks -srckeystore puppetdb.p12
-srcstoretype PKCS12 -alias puppetdb.example.com
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
```

You can validate this was correct:

```
# keytool -list -keystore keystore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

puppetdb.example.com, Mar 30, 2012, PrivateKeyEntry,
Certificate fingerprint (MD5): 7E:2A:B4:4D:1E:6D:D1:70:A9:E7:20:0D:9D:41:F3:B9
```

Compare to the certificate's fingerprint on the CA puppet master:

```
$ sudo puppet cert fingerprint puppetdb.example.com --digest=md5
MD5 Fingerprint=7E:2A:B4:4D:1E:6D:D1:70:A9:E7:20:0D:9D:41:F3:B9
```

On the PuppetDB Server: Move the Keystore and Truststore

Take the truststore and keystore you generated in the preceding steps and copy them to a permanent home. These instructions will assume you are using `/etc/puppetdb/ssl`.

Change the files' ownership to the user PuppetDB will run as, and ensure that only that user can read the files:

```
$ sudo chown puppetdb:puppetdb /etc/puppetdb/ssl/truststore.jks
/etc/puppetdb/ssl/keystore.jks
$ sudo chmod 400 /etc/puppetdb/ssl/truststore.jks
/etc/puppetdb/ssl/keystore.jks
```

You can now safely delete the temporary copies of the keystore, truststore, CA certificate, PuppetDB certificate and private key. These can be retrieved or recreated using the original copies stored on the CA puppet master.

You should now configure HTTPS in PuppetDB's config file(s); see below.

Step 4: Configure HTTPS

In your PuppetDB configuration file(s), edit the `[jetty]` section. If you installed from source, edit `/etc/puppetdb/conf.d/jetty.ini`; if you are running from source, edit the config file you chose.

The `[jetty]` section should contain the following, with your PuppetDB server's hostname and desired ports:

```
[jetty]
# Optional settings:
host = puppetdb.example.com
port = 8080
# Required settings:
ssl-host = puppetdb.example.com
ssl-port = 8081
keystore = /etc/puppetdb/ssl/keystore.jks
truststore = /etc/puppetdb/ssl/truststore.jks
key-password = <password used when creating the keystore>
trust-password = <password used when creating the truststore>
```

If you ran the SSL configuration script, the password will be in `/etc/puppetdb/ssl/puppetdb_keystore_pw.txt`. Use this for both the `key-password` and the `trust-password`.

If you don't want to do unsecured HTTP at all, you can omit the `host` and `port` settings. However, this may limit your ability to use PuppetDB for other purposes, including viewing its performance dashboard. A reasonable compromise is to set `host` to `localhost`, so that unsecured traffic is only allowed from the local box; tunnels can then be used to gain access to the performance dashboard.

Step 5: Configure Database

If this is a production deployment, you should confirm and configure your database settings:

- Deployments of 100 nodes or fewer can continue to use the default built-in database backend, but should increase PuppetDB's maximum heap size to at least 1 GB.
- Large deployments should set up a PostgreSQL server and configure PuppetDB to use it. You may also need to adjust the maximum heap size.

You can change PuppetDB's database at any time, but note that changing the database does not migrate PuppetDB's data, so the new database will be empty. However, as this data is automatically generated many times a day, PuppetDB should recover in a relatively short period of time.

Step 6: Start the PuppetDB Service

If you installed PuppetDB from source, you can start PuppetDB by running the following:

```
$ sudo /etc/init.d/puppetdb start
```

And if Puppet is installed, you can permanently enable PuppetDB by running:

```
$ sudo puppet resource service puppetdb ensure=running enable=true
```

If you are running PuppetDB from source, you should start it as follows:

```
# From the directory in which PuppetDB's source is stored:
$ lein run services -c /path/to/config.ini
```

PuppetDB is now fully functional and ready to receive catalogs and facts from any number of puppet master servers.

Finish: Connect Puppet to PuppetDB

You should now configure your puppet master(s) to connect to PuppetDB.

If you use a standalone Puppet site, you should configure every node to connect to PuppetDB.

PuppetDB 1 » Using PuppetDB

Currently, the main use for PuppetDB is to enable advanced features in Puppet. We expect additional applications to be built on PuppetDB as it becomes more widespread.

If you wish to build applications on PuppetDB, see the navigation sidebar for links to the API spec.

Checking Node Status

The PuppetDB plugins installed on your puppet master(s) include a `status` action for the `node` face. On your puppet master, run:

```
$ sudo puppet node status <node>
```

where is the name of the node you wish to investigate. This will tell you whether the node is active, when its last catalog was submitted, and when its last facts were submitted.

Using Exported Resources

PuppetDB lets you use exported resources, which allows your nodes to publish information for use by other nodes.

See here for more about using exported resources.

Using the Inventory Service

PuppetDB provides better performance for Puppet's inventory service.

See here for more about using the inventory service and building applications on it. If you are using Puppet Enterprise's console, or Puppet Dashboard with inventory support turned on, you will not need to change your configuration — PuppetDB will become the source of inventory information as soon as the puppet master is connected to it.

PuppetDB 1 » Maintaining and Tuning

PuppetDB requires a relatively small amount of maintenance and tuning. You should become familiar with the following occasional tasks:

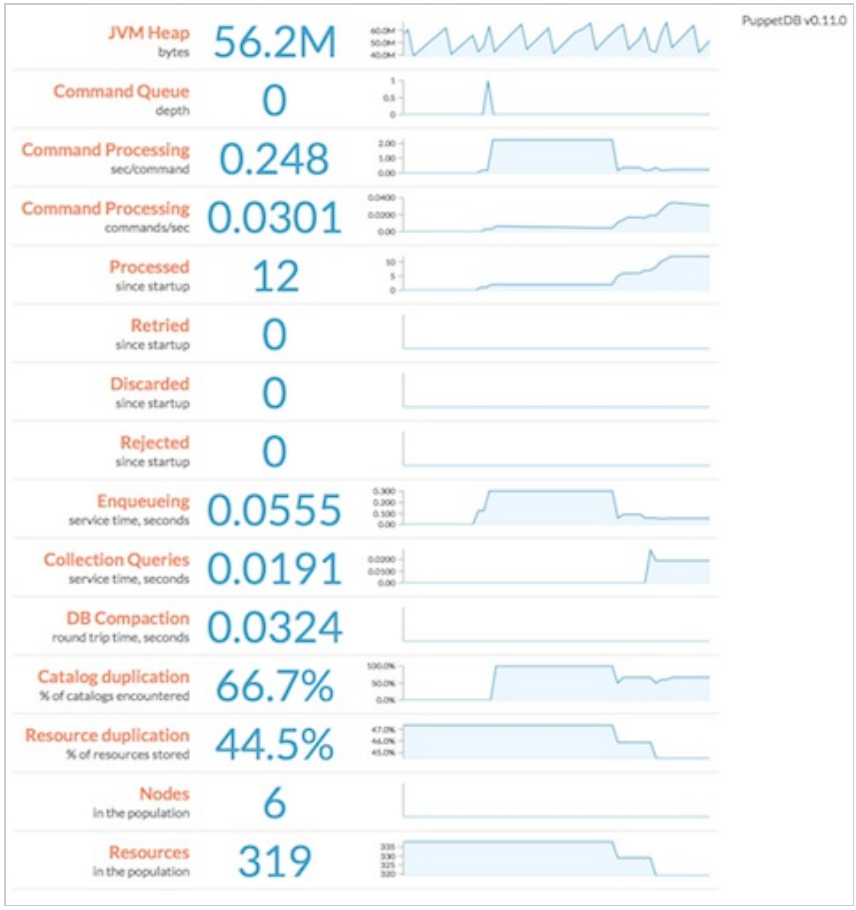
Monitor the Performance Dashboard

Once you have PuppetDB running, visit the following URL, substituting in the name and port of your PuppetDB server:

`http://puppetdb.example.com:8080/dashboard/index.html`

> Note: You may need to edit PuppetDB's HTTP configuration first, changing the `host` setting to the server's externally-accessible hostname. When you do this, you should also configure your firewall to control access to PuppetDB's cleartext HTTP port.

PuppetDB uses this page to display a web-based dashboard with performance information and metrics, including its memory use, queue depth, command processing metrics, duplication rate, and query stats. It displays min/max/median of each metric over a configurable duration, as well as an animated SVG "sparkline" (a simple line chart that shows general variation). It also displays the current version of PuppetDB and checks for updates, showing a link to the latest package if your deployment is out of date.



You can use the following URL parameters to change the attributes of the dashboard:

- `width` = width of each sparkline, in pixels

- `height` = height of each sparkline, in pixels
- `nHistorical` = how many historical data points to use in each sparkline
- `pollingInterval` = how often to poll PuppetDB for updates, in milliseconds

E.g.: `http://puppetdb.example.com:8080/dashboard/index.html?height=240&pollingInterval=1000`

Deactivate Decommissioned Nodes

When you remove a node from your Puppet deployment, it should be marked as deactivated in PuppetDB. This will ensure that any resources exported by that node will stop appearing in the catalogs served to the remaining agent nodes.

- PuppetDB can automatically deactivate nodes that haven't checked in recently. To enable this, set the `node-ttl-days` setting.
- If you prefer to manually deactivate nodes, use the following command on your puppet master:

```
$ sudo puppet node deactivate <node> [<node> ...]
```

- Any deactivated node will be reactivated if PuppetDB receives new catalogs or facts for it.

Although deactivated nodes will be excluded from storeconfigs queries, their data is still preserved.

> Note: Deactivating a node does not remove (e.g. `ensure => absent`) exported resources from other systems; it only stops managing those resources. If you want to actively destroy resources from deactivated nodes, you will probably need to purge that resource type using the `resources` metatype. Note that some types cannot be purged (e.g. ssh authorized keys), and several others usually should not be purged (e.g. users).

View the Log

PuppetDB's log file lives at `/var/log/pe-puppetdb/pe-puppetdb.log` (for PE users) or `/var/log/puppetdb/puppetdb.log` (for open source users). Check the log when you need to confirm that PuppetDB is working correctly or to troubleshoot visible malfunctions. If you have changed the logging settings, examine the log4j.properties file to find the log.

The PuppetDB packages install a logrotate job in `/etc/logrotate.d/puppetdb`, which will keep the log from becoming too large.

Tune the Max Heap Size

Although we provide rule-of-thumb memory recommendations, PuppetDB's RAM usage depends on several factors, so memory needs will vary depending on the number of nodes, frequency of Puppet runs, and amount of managed resources. 1000 nodes that check in once a day will require much less memory than if they check in every 30 minutes.

So the best way to manage PuppetDB's max heap size is to estimate a ballpark figure, then monitor the performance dashboard and increase the heap size if the "JVM Heap" metric keeps approaching the maximum. You may need to revisit your memory needs whenever your site grows substantially.

The good news is that memory starvation is actually not very destructive. It will cause `OutOfMemoryError` exceptions to appear in the log, but you can restart PuppetDB with a larger memory allocation and it'll pick up where it left off — any requests successfully queued up in PuppetDB will get processed.

Tune the Number of Threads

When viewing the performance dashboard, note the MQ depth. If it is rising and you have CPU cores to spare, increasing the number of threads may help process the backlog faster.

If you are saturating your CPU, we recommend lowering the number of threads. This prevents other PuppetDB subsystems (such as the web server, or the MQ itself) from being starved of resources and can actually increase throughput.

Redo SSL Setup After Changing Certificates

If you've recently changed the certificates in use by the PuppetDB server, you'll also need to update the SSL configuration for PuppetDB itself.

If you've installed PuppetDB from Puppet Labs packages, you can simply re-run the `puppetdb-ssl-setup` script. Otherwise, you'll need to again perform all the SSL configuration steps outlined in the installation instructions.

PuppetDB 1 » Scaling Recommendations

Since PuppetDB will be a critical component of your Puppet deployment (that is, agent nodes will be unable to request catalogs if it goes down), you should make sure it can handle your site's load and is resilient against failures.

As with scaling any service, there are several possible performance and reliability bottlenecks which can be dealt with in turn as they become problems.

Bottleneck: Database Performance

Database Backend

PuppetDB has two available database backends:

- Embedded HSQLDB
- PostgreSQL

The embedded database works with no additional daemons or setup beyond installation, but is only suitable for up to about 100 Puppet nodes. It also requires a signifiacntly larger Java heap.

You can increase performance by setting up a PostgreSQL server and switching PuppetDB to the PostgreSQL backend.

PostgreSQL Speed and Availability

Using the PostgreSQL backend, PuppetDB will be limited by the performance of your Postgres server. You can increase performance by making sure your DB server has an extremely fast disk,

plenty of RAM, a fast processor, and a fast network connection to your PuppetDB server. You may also need to look into database clustering and load balancing.

Database administration is beyond the scope of this manual, but the following links may be helpful:

- High Availability, Load Balancing, and Replication, from the PostgreSQL manual
- Replication, Clustering, and Connection Pooling, from the PostgreSQL wiki

Bottleneck: Java Heap Size

PuppetDB is limited by the amount of memory available to it, which is set in the init script's config file. If it runs out of memory, it will start logging `OutOfMemoryError` exceptions and delaying command processing. Unlike many of the bottlenecks listed here, this one is fairly binary: PuppetDB either has enough memory to function under its load, or it doesn't. The exact amount needed will depend on the DB backend, the number of nodes, the similarity of the nodes, the complexity of each node's catalog, and how often the nodes check in.

Initial Memory Recommendations

Use one of the following rules of thumb to choose an initial heap size; afterwards, watch the performance dashboard and adjust the heap if necessary.

- If you are using PostgreSQL, allocate 128 MB of memory as a base, plus 1 MB for each Puppet node in your infrastructure.
- If you are using the embedded database, allocate at least 1 GB of heap.

Bottleneck: Node Checkin Interval

The more frequently your Puppet nodes check in, the heavier the load on your PuppetDB server.

You can reduce the need for higher performance by changing the `runinterval` setting in every Puppet node's puppet.conf file. (Or, if running puppet agent from cron, by changing the frequency of the cron task.)

The frequency with which nodes should check in will depend on your site's policies and expectations — this is just as much a cultural decision as it is a technical one. A possible compromise is to use a wider default checkin interval, but implement MCollective's `puppetd` plugin to trigger immediate runs when needed.

Bottleneck: CPU Cores and Number of Worker Threads

PuppetDB can take advantage of multiple CPU cores to handle the commands in its queue. Each core can run a worker thread; by default, PuppetDB will use half of the cores in its machine.

You can increase performance by running PuppetDB on a machine with many CPU cores and then tuning the number of worker threads:

- More threads will allow PuppetDB to keep up with more incoming commands per minute. Watch the queue depth in the performance dashboard to see whether you need more threads.
- Too many worker threads can potentially starve the message queue and web server of resources, which will prevent incoming commands from entering the queue in a timely fashion. Watch your server's CPU usage to see whether the cores are saturated.

Bottleneck: Single Point of Failure

Although a single PuppetDB and PostgreSQL server probably can handle all of the load at the site, you may want to run multiple servers for the sake of resilience and redundancy. To configure high-availability PuppetDB, you should:

- Run multiple instances of PuppetDB on multiple servers, and use a reverse proxy or load balancer to distribute traffic between them.
- Configure multiple PostgreSQL servers for high availability or clustering. More information is available at the PostgreSQL manual and the PostgreSQL wiki.
- Configure every PuppetDB instance to use the same PostgreSQL database. (In the case of clustered Postgres servers, they may be speaking to different machines, but conceptually they should all be writing to one database.)

Bottleneck: SSL Performance

PuppetDB uses its own embedded SSL processing, which is usually not a performance problem. However, truly large deployments will be able to squeeze out more performance by terminating SSL with Apache or Nginx instead. If you are using multiple PuppetDB servers behind a reverse proxy, we recommend terminating SSL at the proxy server.

Instructions for configuring external SSL termination are currently beyond the scope of this manual. If your site is big enough for this to be necessary, you have probably done it with several other services before.

PuppetDB 1 » Configuration

Summary

PuppetDB has three main groups of settings:

- The init script's configuration file, which sets the Java heap size and the location of PuppetDB's main config file
- Logging settings, which go in the log4j.properties file and can be changed without restarting PuppetDB
- All other settings, which go in PuppetDB's configuration file(s) and take effect after the service is restarted

Init Script Config File

If you installed PuppetDB from packages or used the `rake install` installation method, an init script was created for PuppetDB. This script has its own configuration file, whose location varies by platform and by package:

OS and Package	File
Redhat-like (open source)	`/etc/sysconfig/puppetdb`
Redhat-like (PE)	`/etc/sysconfig/pe-puppetdb`
Debian/Ubuntu (open source)	`/etc/default/puppetdb`
Debian/Ubuntu (PE)	`/etc/default/pe-puppetb`

In this file, you can change the following settings:

`JAVA_BIN`
The location of the Java binary.
`JAVA_ARGS`
Command line options for the Java binary, most notably the `-Xmx` (max heap size) flag.
`USER`
The user PuppetDB should be running as.
`INSTALL_DIR`
The directory into which PuppetDB is installed.
`CONFIG`
The location of the PuppetDB config file, which may be a single file or a directory of .ini files.

CONFIGURING THE JAVA HEAP SIZE

To change the JVM heap size for PuppetDB, edit the init script config file by setting a new value for the `-Xmx` flag in the `JAVA_ARGS` variable.

For example, to cap PuppetDB at 192MB of memory:

```
JAVA_ARGS="-Xmx192m"
```

To use 1GB of memory:

```
JAVA_ARGS="-Xmx1g"
```

Configuring Logging

Logging is configured with a log4j.properties file, whose location is defined with the `logging-config` setting. If you change the log settings while PuppetDB is running, it will apply the new settings without requiring a restart.

See the log4j documentation for more information about logging options.

The PuppetDB Configuration File(s)

PuppetDB is configured using an INI-style config format with several `[sections]`. This is very similar to the format used by Puppet.

Whenever you change PuppetDB's configuration settings, you must restart the service for the changes to take effect.

You can change the location of the main config file in the init script config file. This location can point to a single configuration file or a directory of .ini files. If you specify a directory (conf.d style), PuppetDB will merge the .ini files in alphabetical order.

If you've installed PuppetDB from a package, by default it will use the conf.d config style. The default config directory is `/etc/puppetdb/conf.d` (or `/etc/puppetlabs/puppetdb/conf.d` for Puppet Enterprise). If you're running from source, you may use the `-c` command-line argument to specify your config file or directory.

An example configuration file:

```
[global]
vardir = /var/lib/puppetdb
logging-config = /var/lib/puppetdb/log4j.properties
```

```
resource-query-limit = 20000

[database]
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //localhost:5432/puppetdb

[jetty]
port = 8080
```

`[global]` Settings

The `[global]` section is used to configure application-wide behavior.

`vardir`

This defines the parent directory for the MQ's data directory. Also, if a database isn't specified, the default database's files will be stored in `<vardir>/db`. The directory must exist and be writable by the PuppetDB user in order for the application to run.

`logging-config`

This describes the full path to a log4j.properties file. Covering all the options available for configuring log4j is outside the scope of this document; see the aforementioned link for exhaustive information.

If this setting isn't provided, PuppetDB defaults to logging at INFO level to standard out.

If you installed from packages, PuppetDB will use the log4j.properties file in the `/etc/puppetdb/` or `/etc/puppetlabs/puppetdb` directory. Otherwise, you can find an example file in the `ext` directory of the source.

You can edit the logging configuration file while PuppetDB is running, and it will automatically react to changes after a few seconds.

`resource-query-limit`

The maximum number of legal results that a resource query can return. If you issue a query that would result in more results than this value, the query will simply return an error. (This can be used to prevent accidental queries that would yield huge numbers of results from consuming undesirable amounts of resources on the server.)

The default value is 20000.

`[database]` Settings

The `[database]` section configures PuppetDB's database settings.

PuppetDB can use either a built-in HSQLDB database or a PostgreSQL database. If no database information is supplied, an HSQLDB database at `<vardir>/db` will be used.

> FAQ: Why no MySQL or Oracle support?
>
> MySQL lacks several features that PuppetDB relies on; the most notable is recursive queries. We have no plans to ever support MySQL.

> Depending on demand, Oracle support may be forthcoming in a future version of PuppetDB. This hasn't been decided yet.

Using Built-in HSQLDB

To use an HSQLDB database at the default `<vardir>/db`, you can simply remove all database settings. To configure the DB for a different location, put the following in the `[database]` section:

```
classname = org.hsqldb.jdbcDriver
subprotocol = hsqldb
subname = file:</PATH/TO/DB>;hsqldb.tx=mvcc;sql.syntax_pgs=true
```

Replace `</PATH/TO/DB>` with the filesystem location in which you'd like to persist the database.

Do not use the `username` or `password` settings.

Using PostgreSQL

Before using the PostgreSQL backend, you must set up a PostgreSQL server, ensure that it will accept incoming connections, create a user for PuppetDB to use when connecting, and create a database for PuppetDB. Completely configuring PostgreSQL is beyond the scope of this manual, but if you are logged in as root on a running Postgres server, you can create a user and database as follows:

```
$ sudo -u postgres sh
$ createuser -DRSP puppetdb
$ createdb -O puppetdb puppetdb
$ exit
```

Ensure you can log in by running:

```
$ psql -h localhost puppetdb puppetdb
```

To configure PuppetDB to use this database, put the following in the `[database]` section:

```
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //<HOST>:<PORT>/<DATABASE>
username = <USERNAME>
password = <PASSWORD>
```

Replace `<HOST>` with the DB server's hostname. Replace `<PORT>` with the port on which PostgreSQL is listening. Replace `<DATABASE>` with the name of the database you've created for use with PuppetDB.

It's possible to use SSL to protect connections to the database. The PostgreSQL JDBC docs explain how to do this. Be sure to add `?ssl=true` to the `subname` setting:

```
subname = //<host>:<port>/<database>?ssl=true
```

`gc-interval`

This controls how often, in minutes, to compact the database. The compaction process reclaims space and deletes unnecessary rows. If not supplied, the default is every 60 minutes.

`node-ttl-days`

This sets the number of days with no activity (no new catalogs, facts, etc) before a node will be auto-deactivated. Nodes will be checked for staleness every `gc-interval` minutes. Manual deactivation will continue to work as always.

If unset, auto-deactivation of nodes is disabled.

`log-slow-statements`

This sets the number of seconds before an SQL query is considered "slow." Slow SQL queries are logged as warnings, to assist in debugging and tuning. Note PuppetDB does not interrupt slow queries; it simply reports them after they complete.

The default value is 10 seconds. A value of 0 will disable logging of slow queries.

`classname`

This sets the JDBC class to use. Set this to:

- `org.hsqldb.jdbcDriver` when using the embedded database
- `org.postgresql.Driver` when using PostgreSQL

`subprotocol`

Set this to:

- `hsqldb` when using the embedded database
- `postgresql` when using PostgreSQL

`subname`

This describes where to find the database. Set this to:□

- `file:</PATH/TO/DB>;hsqldb.tx=mvcc;sql.syntax_pgs=true` when using the embedded database, replacing `</PATH/TO/DB>` with a local filesystem path□
- `//<HOST>:<PORT>/<DATABASE>` when using PostgreSQL, replacing `<HOST>` with the DB server's hostname, `<PORT>` with the port on which PostgreSQL is listening, and `<DATABASE>` with the name of the database
  - Append `?ssl=true` to this if your PostgreSQL server is using SSL.

`username`

This is the username to use when connecting. Only used with PostgreSQL.

`password`

This is the password to use when connecting. Only used with PostgreSQL.

`[command-processing]` Settings

The `[command-processing]` section configures the command-processing subsystem.

Every change to PuppetDB's data stores arrives via commands that are inserted into a message queue (MQ). Command processor threads pull items off of that queue, persisting those changes.

`threads`

This defines how many command processing threads to use. Each thread can process a single command at a time. The number of threads can be tuned based on what you see in the performance dashboard.

This setting defaults to half the number of cores in your system.

`[jetty]` (HTTP) Settings

The `[jetty]` section configures HTTP for PuppetDB.

`host`

This sets the hostname to listen on for unencrypted HTTP traffic. If not supplied, we bind to `localhost`, which will reject connections from anywhere but the PuppetDB server itself. To listen on all available interfaces, use `0.0.0.0`.

> Note: Unencrypted HTTP is the only way to view the performance dashboard, since PuppetDB uses host verification for SSL. However, it can also be used to make any call to PuppetDB's API, including inserting exported resources and retrieving arbitrary data about your Puppet-managed nodes. If you enable cleartext HTTP, you MUST configure your firewall to protect unverified access to PuppetDB.

`port`

This sets what port to use for unencrypted HTTP traffic. If not supplied, we won't listen for unencrypted traffic at all.

`ssl-host`

This sets the hostname to listen on for encrypted HTTPS traffic. If not supplied, we bind to `localhost`. To listen on all available interfaces, use `0.0.0.0`.

`ssl-port`

This sets the port to use for encrypted HTTPS traffic. If not supplied, we won't listen for encrypted traffic at all.

`keystore`

This sets the path to a Java keystore file containing the key and certificate to be used for HTTPS.

`key-password`

This sets the passphrase to use for unlocking the keystore file.

`truststore`

This describes the path to a Java keystore file containing the CA certificate(s) for your puppet

infrastructure.

`trust-password`

This sets the passphrase to use for unlocking the truststore file.

`certificate-whitelist`

Optional. This describes the path to a file that contains a list of certificate names, one per line. Incoming HTTPS requests will have their certificates validated against this list of names and only those with an exact matching entry will be allowed through. (For a puppet master, this compares against the value of the `certname` setting, rather than the `dns_alt_names` setting.)

If not supplied, PuppetDB uses standard HTTPS without any additional authorization. All HTTPS clients must still supply valid, verifiable SSL client certificates.

`[repl]` Settings

The `[repl]` section configures remote runtime modification.

Enabling a remote REPL allows you to manipulate the behavior of PuppetDB at runtime. This should only be done for debugging purposes, and is thus disabled by default. An example configuration stanza:

```
[repl]
enabled = true
type = nrepl
port = 8081
```

`enabled`

Set to `true` to enable the REPL. Defaults to false.

`type`

Either `nrepl` or `swank`.

The nrepl repl type opens up a socket you can connect to via telnet.

The swank type allows emacs' clojure-mode to connect directly to a running PuppetDB instance by using `M-x slime-connect`. This is much more user-friendly than telnet.

`port`

The port to use for the REPL.

PuppetDB 1 » Debugging with the Remote REPL

PuppetDB includes a remote REPL interface, which is disabled by default.

This is mostly of use to developers who know Clojure and are familiar with PuppetDB's code base. It allows you to modify PuppetDB's code on the fly. Most users should never need to use the REPL, and

it should usually be left disabled for security reasons.

Enabling the REPL

To enable the REPL, you must edit PuppetDB's config file to enable it, configure the REPL type, and choose a port:

```
# /etc/puppetdb/conf.d/repl.ini
[repl]
enabled = true
type = nrepl
port = 8081
```

After configuring it, you should restart the PuppetDB service.

Connecting to a Remote REPL

Once PuppetDB is accepting remote REPL connections, you can connect to it and begin issuing low-level debugging commands and Clojure code.

For example, with an nrepl type REPL configured on port 8082:

```
$ telnet localhost 8082
Connected to localhost.
Escape character is '^]'.
;; Clojure 1.3.0
user=> (+ 1 2 3)
6
```

Executing Functions

Within the REPL, you can interactively execute PuppetDB's functions. For example, to manually compact the database:

```
user=> (use 'com.puppetlabs.puppetdb.cli.services)
nil
user=> (use 'com.puppetlabs.puppetdb.scf.storage)
nil
user=> (use 'clojure.java.jdbc)
nil
user=> (with-connection (:database configuration)
         (garbage-collect!))
(0)
```

Redefining Functions

You can also manipulate the running PuppetDB instance by redefining functions on the fly. Let's say that for debugging purposes, you'd like to log every time a catalog is deleted. You can just redefine the existing `delete-catalog!` function dynamically:

```
user=> (ns com.puppetlabs.puppetdb.scf.storage)
nil
com.puppetlabs.puppetdb.scf.storage=>
(def original-delete-catalog! delete-catalog!)
```

```
#'com.puppetlabs.puppetdb.scf.storage/original-delete-catalog!
com.puppetlabs.puppetdb.scf.storage=>
(defn delete-catalog!
  [catalog-hash]
  (log/info (str "Deleting catalog " catalog-hash))
  (original-delete-catalog! catalog-hash))
#'com.puppetlabs.puppetdb.scf.storage/delete-catalog!
```

Now any time that function is called, you'll see a message logged.

Note that any changes you make to the running system are transient; they don't persist between restarts of the service. If you wish to make longer-lived changes to the code, consider running PuppetDB directly from source.

PuppetDB 1 » Spec » Curl Tips

You can use `curl` to directly interact with PuppetDB's REST API. This is useful for testing, prototyping, and quickly fetching arbitrary data.

The instructions below are simplified. For full usage details, see the curl manpage . For additional examples, please see the docs for the individual REST endpoints:

- Facts
- Nodes
- Resources
- Status
- Metrics

Using `curl` From `localhost` (Non-SSL/HTTP)

With its default settings, PuppetDB accepts unsecured HTTP connections at port 8080 on `localhost`. This allows you to SSH into the PuppetDB server and run curl commands without specifying certificate information:

```
curl -H "Accept: application/json" 'http://localhost:8080/facts/<node>'
curl -H "Accept: application/json"
'http://localhost:8080/metrics/mbean/java.lang:type=Memory'
```

If you have allowed unsecured access to other hosts in order to monitor the dashboard, these hosts can also use plain HTTP curl commands.

Using `curl` From Remote Hosts (SSL/HTTPS)

To make secured requests from other hosts, you will need to supply the following via the command line:

- Your site's CA certificate (`--cacert`)
- An SSL certificate signed by your site's Puppet CA (`--cert`)
- The private key for that certificate (`--key`)

Any node managed by puppet agent will already have all of these and you can re-use them for contacting PuppetDB. You can also generate a new cert on the CA puppet master with the `puppet cert generate` command.

> Note: If you have turned on certificate whitelisting, you must make sure to authorize the certificate you are using.

```
curl -H "Accept: application/json"
'https://<your.puppetdb.server>:8081/facts/<node>' --cacert
/etc/puppet/ssl/certs/ca.pem --cert /etc/puppet/ssl/certs/<node>.pem --key
/etc/puppet/ssl/private_keys/<node>.pem
```

Locating Puppet Certificate Files

Locate Puppet's `ssldir` as follows:

```
$ sudo puppet config print ssldir
```

Within this directory:

- The CA certificate is found at `certs/ca.pem`
- The corresponding private key is found at `private_keys/<name>.pem`
- Other certificates are found at `certs/<name>.pem`

Dealing with complex query strings

Many query strings will contain characters like `[` and `]`, which must be URL-encoded. To handle this, you can use `curl`'s `--data-urlencode` option.

If you do this with an endpoint that accepts `GET` requests, you must also use the `-G` or `--get` option. This is because `curl` defaults to `POST` requests when the `--data-urlencode` option is present.

```
curl -G -H "Accept: application/json" 'http://localhost:8080/nodes' --data-
urlencode 'query=["=", ["node", "active"], true]'
```

PuppetDB 1 » Spec » Commands

Commands are used to change PuppetDB's model of a population. Commands are represented by `command objects`, which have the following JSON wire format:

```
{"command": "...",
 "version": 123,
 "payload": <json object>}
```

`command` is a string identifying the command.

`version` is a JSON integer describing what version of the given command you're attempting to invoke.

`payload` must be a valid JSON string of any sort. It's up to an individual handler function to determine how to interpret that object.

The entire command MUST be encoded as UTF-8.

Command submission

Commands are submitted via HTTP to the `/commands/` URL and must conform to the following rules:

- A `POST` is used
- There is a parameter, `payload`, that contains the entire command as outlined above.
- There is an `Accept` header that contains `application/json`.
- The POST body is url-encoded
- The content-type is `x-www-form-urlencoded`.

Optionally, there may be a parameter, `checksum`, that contains a SHA-1 hash of the payload which will be used for verification.

When a command is successfully submitted, the submitter will receive the following:

- A response code of 200
- A content-type of `application/json`
- A response body in the form of a JSON object, containing a single key 'uuid', whose value is a UUID corresponding to the submitted command. This can be used, for example, by clients to correlate submitted commands with server-side logs.

Command Semantics

Commands are processed asynchronously. If PuppetDB returns a 200 when you submit a command, that only indicates that the command has been accepted for processing. There are no guarantees as to when that command will be processed, nor that when it is processed it will be successful.

Commands that fail processing will be stored in files in the "dead letter office", located under the MQ data directory, in `discarded/<command>`. These files contain the command and diagnostic information that may be used to determine why the command failed to be processed.

List of Commands

"replace catalog", version 1

The payload is expected to be a Puppet catalog conforming to the catalog wire format.

"replace facts", version 1

The payload is expected to be a set of Puppet facts conforming to the facts wire format.

"deactivate node", version 1

The payload is expected to be the name of a node, which will be deactivated effective as of the time

the command is processed.

PuppetDB 1 » Spec » Querying Nodes

Nodes can be queried by making an HTTP request to the `/nodes` REST endpoint with a JSON-formatted parameter called `query`.

Query format

- The HTTP method must be `GET`.
- There must be an `Accept` header specifying `application/json`.

The `query` parameter uses a format similar to resource queries.

Only queries against facts and filters based on node activeness are currently supported.

These query terms must be of the form `["fact", "<fact name>"]` or `["node", "active"]`, respectively.

Accepted operators are: `[= > < >= <= and or not]`

Inequality operators are strictly arithmetic, and will ignore any fact values which are not numeric.

Note that nodes which are missing a fact referenced by a `not` query will match the query.

In this example, the query will return active nodes whose kernel is Linux and whose uptime is less than 30 days:

```
["and",
  ["=", ["node", "active"], true],
  ["=", ["fact", "kernel"], "Linux"],
  [">", ["fact", "uptime_days"], 30]]
```

If no `query` parameter is supplied, all nodes will be returned.

Response format

The response is a JSON array of node names that match the predicates, sorted in ascending order:

```
["foo.example.com", "bar.example.com", "baz.example.com"]
```

Example

Using `curl` from localhost:

Retrieving all nodes:

```
curl -H "Accept: application/json" 'http://localhost:8080/nodes'
```

Retrieving all active nodes:

```
curl -G -H "Accept: application/json" 'http://localhost:8080/nodes' --data-
urlencode 'query=["=", ["node", "active"], true]'
```

PuppetDB 1 » Spec » Querying Facts

Querying facts occurs via an HTTP request to the `/facts` REST endpoint.

Query format

Facts are queried by making a request to a URL in the following form:

The HTTP request must conform to the following format:

- The URL requested is `/facts/<node>`
- A `GET` is used.
- There is an `Accept` header containing `application/json`.

The supplied `<node>` path component indicates the certname for which facts should be retrieved.

Response format

```
{"name": "<node>",
 "facts": {
     "<fact name>": "<fact value>",
     "<fact name>": "<fact value>",
     ...
    }
}
```

If no facts are known for the supplied node, an HTTP 404 is returned.

Example

Using `curl` from localhost:

```
curl -H "Accept: application/json" 'http://localhost:8080/facts/<node>'
```

Where `<node>` is the name of the node from which you wish to retrieve facts.

PuppetDB 1 » Spec » Querying Resources

Resources are queried via an HTTP request to the `/resources` REST endpoint.

Query format

Queries for resources must conform to the following format:

- A `GET` is used.

- There is a single parameter, `query`.
- There is an `Accept` header containing `application/json`.
- The `query` parameter is a JSON array of query predicates, in prefix form, conforming to the format described below.

The `query` parameter adheres to the following grammar:

```
query: [ {type} {query}+ ] | [ {match} {field} {value} ]
field:  string | [ string+ ]
value:  string
type:   "or" | "and" | "not"
match:  "="
```

`field` strings may be any of the following:

`tag`
: a case-insensitive tag on the resource

`["node", "name"]`
: the name of the node associated with the resource

`["node", "active"]`
: `true` if the node has not been deactivated, `false` if it has

`["parameter", "<parameter name>"]`
: a parameter of the resource

`type`
: the resource type

`title`
: the resource title

`exported`
: whether or not the resource is exported

`sourcefile`
: the manifest file where the resource was declared

`sourceline`
: the line of the manifest in which the resource was declared

For example, the JSON query structure for file resources, tagged “magical”, and present on any active host except for “example.local” would be:

```
["and", ["not", ["=", ["node", "name"], "example.local"]],
        ["=", ["node", "active"], true],
        ["=", "type", "File"],
        ["=", "tag",  "magical"],
        ["=", ["parameter", "ensure"], "enabled"]]
```

The following conditionals for type behaviors are defined:

`or`
: If any condition is true, the result is true.

`and`
: If all conditions are true, the result is true.

`not`
: If none of the conditions are true, the result is true.

The following match operator behaviors are defined:

`=`
: Exact string equality of the field and the value.

Response format

An array of zero or more resource objects, with each object in the following form:

```
{"certname":   "the certname of the associated host",
 "resource":   "the resource's unique hash",
 "type":       "File",
 "title":      "/etc/hosts",
 "exported":   "true",
 "tags":       ["foo", "bar"],
 "sourcefile": "/etc/puppet/manifests/site.pp",
 "sourceline": "1",
 "parameters": {<parameter>: <value>,
                <parameter>: <value>,
                ...}}
```

Example

Using `curl` from localhost:

Retrieving the resource `File['/etc/ipsec.conf']`:

```
curl -G -H "Accept: application/json" 'http://localhost:8080/resources' --data-
urlencode 'query=["and", ["=", "type", "File"], ["=", "title",
"/etc/ipsec.conf"]]'
```

PuppetDB 1 » Spec » Querying Status

Query Format

Node status can be queried by making an HTTP GET request to `/status/nodes/<node>`, specifying that the request accepts JSON.

Response Format

Node status information will be returned in a JSON hash of the form:

```
{"name": <node>,
 "deactivated": <timestamp>,
 "catalog_timestamp": <timestamp>,
 "facts_timestamp": <timestamp>}
```

If the node is active, “deactivated” will be null. If a catalog or facts are not present, the corresponding timestamps will be null.

If no information is known about the node, the result will be a 404 with a JSON hash containing an “error” key with a message indicating such.

Example

Using `curl` from localhost:

```
curl -H "Accept: application/json" 'http://localhost:8080/status/nodes/<node>'
```

Where is the name of the node whose status you wish to check.

PuppetDB 1 » Spec » Querying Metrics

Querying PuppetDB metrics is accomplished by making an HTTP request to paths under the `/metrics` REST endpoint.

Listing available metrics

Request format

To get a list of all available metric names:

- Request `/metrics/mbeans`.
- Use a `GET` request.
- Provide an `Accept` header containing `application/json`.

Response format

Responses return a JSON Object mapping a string to a string:

- The key is the name of a valid MBean
- The value is a URI to use for requesting that MBean's attributes

Retrieving a specific metric□

Request format

To get the attributes of a particular metric:

- Request `/metrics/mbean/<name>`, where `<name>` is something that was returned in the list of available metrics specified above.□
- Use a `GET` request.
- Provide an `Accept` header containing `application/json`.

Response format

Responses return a JSON Object mapping strings to (strings/numbers/booleans).

Useful metrics

Population metrics

- `com.puppetlabs.puppetdb.query.population:type=default,name=num-nodes`: The number of nodes in your population.
- `com.puppetlabs.puppetdb.query.population:type=default,name=num-resources`: The number of resources in your population.
- `com.puppetlabs.puppetdb.query.population:type=default,name=avg-resources-per-node`: The average number of resources per node in your population.
- `com.puppetlabs.puppetdb.query.population:type=default,name=pct-resource-dupes`: The percentage of resources that exist on more than one node.

Database metrics

- `com.jolbox.bonecp:type=BoneCP`: Database connection pool metrics. How long it takes to get a free connection, average execution times, number of free connections, etc.

Command-processing metrics

Each of the following metrics is available for each command supported in PuppetDB. In the below list of metrics, `<name>` should be substituted with a command specifier. Example `<name>`s you can use include:

- `global`: Aggregate stats for all commands
- `replace catalog.1`: Stats for catalog replacement
- `replace facts.1`: Stats for facts replacement
- `deactivate node.1`: Stats for node deactivation

Other than `global`, all command specifiers are of the form `<command>.<version>`. As we version commands, you'll be able to get statistics for each version independently.

Metrics available for each command:

- `com.puppetlabs.puppetdb.command:type=<name>,name=discarded`: stats about commands we've discarded (we've retried them as many times as we can, to no avail)
- `com.puppetlabs.puppetdb.command:type=<name>,name=fatal`: stats about commands we failed to process.
- `com.puppetlabs.puppetdb.command:type=<name>,name=processed`: stats about commands we've successfully processed
- `com.puppetlabs.puppetdb.command:type=<name>,name=processing-time`: stats about how long it takes to process commands
- `com.puppetlabs.puppetdb.command:type=<name>,name=retried`: stats about commands that have been submitted for retry (due to transient errors)

HTTP metrics

Each of the following metrics is available for each HTTP endpoint. In the below list of metrics, `<name>` should be substituted with a REST endpoint name. Example `<name>`s you can use include:

- `commands`: Stats relating to the command processing REST endpoint. The PuppetDB terminus in Puppet talks to this endpoint to submit new catalogs, facts, etc.
- `metrics`: Stats relating to the metrics REST endpoint. This is the endpoint you're reading about right now!
- `facts`: Stats relating to fact querying. This is the endpoint used by the puppetmaster for inventory service queries.
- `resources`: Stats relating to resource querying. This is the endpoint used when collecting exported resources.

In addition to customizing `<name>`, the following metrics are available for each HTTP status code (`<status code>`). For example, you can see the stats for all `200` responses for the `resources` endpoint. This allows you to see, per endpoint and per response, independent counters and statistics.

- `com.puppetlabs.puppetdb.http.server:type=<name>,name=service-time`: stats about how long it takes to service all HTTP requests to this endpoint
- `com.puppetlabs.puppetdb.http.server:type=<name>,name=<status code>`: stats about how often we're returning this response code

Storage metrics

Metrics involving the PuppetDB storage subsystem all begin with the `com.puppetlabs.puppetdb.scf.storage:type=default,name=` prefix. There are a number of metrics concerned with individual storage operations (storing resources, storing edges, etc.). Metrics of particular note include:

- `com.puppetlabs.puppetdb.scf.storage:type=default,name=duplicate-pct`: the percentage of catalogs that PuppetDB determines to be duplicates of existing catalogs.
- `com.puppetlabs.puppetdb.scf.storage:type=default,name=gc-time`: state about how long it takes to do storage compaction

JVM Metrics

- `java.lang:type=Memory`: memory usage statistics
- `java.lang:type=Threading`: stats about JVM threads

MQ Metrics

- `org.apache.activemq:BrokerName=localhost,Type=Queue,Destination=com.puppetlabs.puppetdb.commands`: stats about the command processing queue: queue depth, how long messages remain in the queue, etc.

Example

Using `curl` from localhost:

```
curl -H "Accept: application/json"
'http://localhost:8080/metrics/mbean/java.lang:type=Memory'
```

PuppetDB 1 » Spec » Catalog Wire Format, Version 1

PuppetDB receives catalogs from puppet masters in the following wire format. This format is subtly different from the internal format used by Puppet so catalogs are converted by the PuppetDB terminus plugins before they are sent. See below for the justification for this separate format.

Catalog Interchange Format

Version

This is version 1 of the catalog interchange format, which is used by PuppetDB 1 (and all pre-1.0 releases).

Encoding

The entire catalog is serialized as JSON, which requires strict UTF-8 encoding. Unless otherwise

noted, null is not allowed anywhere in the catalog.

Main Data Type: Catalog

A catalog is a JSON object with two keys: `"metadata"` and `"data"`.

```
{"metadata": {
    "type": "catalog",
    "version": 1
    }
 "data": {
    "name": <string>,
    "version": <string>,
    "classes":
        [<string>, <string>, ...],
    "tags":
        [<string>, <string>, ...],
    "edges":
        [<edge>, <edge>, ...],
    "resources":
        [<resource>, <resource>, ...]
    }
}
```

The value of the `"metadata"` key must be `{ "type": "catalog", "version": 1 }` — no other value is valid for this version of the format.

The value of the `"data"` key must be a JSON object with six keys: `"name"`, `"version"`, `"classes"`, `"tags"`, `"edges"`, and `"resources"`. Each of the keys is mandatory, although values that are lists may be empty lists. The value of each key in the data object is as follows:

`"name"`
String. The name of the node for which the catalog was compiled.

`"version"`
String. An arbitrary string that uniquely identifies this specific catalog across time for a single node. This is controlled by Puppet's `config_version` setting and is usually the seconds elapsed since the epoch.

`"classes"`
List of strings. This key has no documented use and exists for unknown reasons. It is a complete list of the classes contained in the catalog, but is not used when searching for information about classes via the resource query API.

`"tags"`

> Deprecated: This key is slated for removal in a future version of the catalog format.

List of strings. This key has no current documented use and only exists for historical reasons. It is a list of a subset of the tags that exist on resources in the catalog, but it is not guaranteed to be complete.

`"edges"`
List of `<edge>` objects. Every relationship between any two resources in the catalog, which may have been made with chaining arrows, metaparameters, or the `require` function.

> Notes:
>
> - "Autorequire" relationships are not currently encoded in the catalog.

- This key is significantly different from its equivalent in Puppet's internal catalog format, which only encodes containment edges.

`"resources"`
List of `<resource>` objects. Contains every resource in the catalog.

Data Type: `<string>`

A JSON string. Because the catalog is UTF-8, these must also be UTF-8.

Data Type: `<integer>`

A JSON int.

Data Type: `<boolean>`

A JSON boolean.

Data Type: `<edge>`

A JSON object of the following form, which represents a relationship between two resources:

```
{"source": <resource-spec>,
 "target": <resource-spec>,
 "relationship": <relationship>}
```

All edges are normalized so that the "source" resource is managed before the "target" resource. To do this, the Puppet language's "require" and "subscribe" relationship types are munged into "required-by" and "subscription-of" when they are converted into edges.

The keys of an edge are `source`, `target`, and `relationship`, all of which are required.

`source`
A `<resource-spec>`. The resource which should be managed first.
`target`
A `<resource-spec>`. The resource which should be managed second.
`relationship`
A `<relationship>`. The way the two resources are related.

Data Type: `<resource-spec>`

(Synonym: `<resource-hash>`.)

The JSON representation of a resource reference (single-resource kind). An object of the following form:

```
{"type": <string>,
 "title": <string>}
```

The resource named by a resource-spec must exist in the catalog's `"resources"` list. Note also that the title must be the resource's actual title, rather than an alias or name/namevar.

Data Type: `<relationship>`

One of the following exact strings, when used in the `relationship` key of an `<edge>` object:

- `contains`
- `before`
- `required-by`
- `notifies`
- `subscription-of`

Note: Regardless of the relationship type, the "source" resource is always managed before the "target" resource. This means that, functionally speaking, `required-by` is a synonym of `before` and `subscription-of` is a synonym of `notifies`. In this catalog format, the different relationship types preserve information about the origin of the relationship.

String	Relationship Type	Origin of Relationship
`contains`	containment	Class or defined type containment
`before`	ordering	`before` metaparam on source, or `->` chaining arrow
`required-by`	ordering	`require` metaparam on target, or `require` function
`notifies`	ordering w/ notification	`notify` metaparam on source, or `~>` chaining arrow
`subscription-of`	ordering w/ notification	`subscribe` metaparam on target

Data Type: `<resource>`

A JSON object of the following form, which represents a Puppet resource:

```
{"type": <string>,
 "title": <string>,
 "aliases": [<string>, <string>, ...],
 "exported": <boolean>,
 "file": <string>,
 "line": <string>,
 "tags": [<string>, <string>, ...],
 "parameters": {<string>: <JSON object>,
                <string>: <JSON object>,
                ...}
}
```

The eight keys in a resource object are `type`, `title`, `aliases`, `exported`, `file`, `line`, `tags` and `parameters`. All of them are required.

`type`
String. The type of the resource, capitalized. (E.g. `File`, `Service`, `Class`, `Apache::Vhost`.) Note that every segment must be capitalized if the type includes a namespace separator (`::`).

`title`
String. The title of the resource.

`aliases`
List of strings. Includes every alias for the resource, including the value of its name/namevar and any extra names added with the `"alias"` metaparameter.

`exported`
Boolean. Whether or not this is an exported resource.

`file`

String. The manifest file in which the resource definition is located.

`line`

Positive integer. The line (of the containing manifest file) at which the resource definition can be found.

`tags`

List of strings. Includes every tag the resource has. This is a normalized superset of the value of the resource's `tag` attribute.

`parameters`

JSON object. Includes all of the resource's attributes and their associated values. The value of an attribute may be any JSON data type, but Puppet will only provide booleans, strings, arrays, and hashes — resource references and numbers in attributes are converted to strings before being inserted into the catalog. Attributes with undef values are not added to the catalog.

Why a new wire format?

Previous Wire Format Shortcomings

There were a number of issues with the built-in JSON wire format used in Puppet prior to PuppetDB:

1. The format isn't actually JSON, it's PSON. This means a catalog may contain non-UTF-8 data. This can present problems for conforming JSON parsers that expect Unicode.
2. Dependency edges aren't represented as first-class entities in the wire format. Instead, dependencies have to be parsed out of resource attributes.
3. Containment edges can point to resources that aren't in the catalog's list of resources. Examples of this include things like `Stage[main]`, or other special classes.
4. There are no (good) provisions for binary data, which can show up in a catalog via use of `generate`, among other functions.
5. Resources can refer to other resources in several ways: by proper name, by alias, by using a type-specific namevar (such as `path` for the file type). None of this is normalized in any way, and consumers of the wire format have to sift through all of this. And for the case of type-specific namevars, it may be impossible for a consumer to reconcile (because the consumer may not have access to puppet source code)

In general, for communication between master and agent, it's useful to have the wire format as stripped-down as possible. But for other consumers, the catalog needs to be precise in its semantics. Otherwise, consumers just end up (poorly) re-coding the catalog-manipulation logic from puppet proper. Hence the need for a wire format that allows consuming code (which may not even originate from puppet) to handle this data.

Differences from Current Wire Format

1. The format is fully documented here.
2. Information that previously had to be deduced by Puppet is now codified inside of the wire format. All possible aliases for a resource are listed as attributes of that resource. The list of edges now contains edges of all types, not just containment edges. And that list of edges is normalized to refer to the `Type` and `Title` of a resource, as opposed to referring to it by any of its aliases.
3. The new format is explicitly versioned. This format is version 1.0.0, unambiguously.
4. Catalogs will be explictly transformed into this format. Currently, the behavior of `#to_pson` is simply expected to "Do The Right Thing" in terms of serialization.

Future Development Goals

1. Binary data support is yet to be developed.
2. The use of a more compact, binary representation of the wire format may be considered. For example, using something like MessagePack, BSON, Thrift, or Protocol Buffers.

PuppetDB 1 » Spec » Facts Wire Format

Format

Facts are represented as JSON. Unless otherwise noted, `null` is not allowed anywhere in the set of facts.

```
{"name": <string>,
 "values": {
     <string>: <string>,
     ...
     }
}
```

The `"name"` key is the certname the facts are associated with.

The `"values"` key points to a JSON Object that represents the set of facts. Each key is the fact name, and the value is the fact value.

Fact names and values MUST be strings.

Encoding

The entire fact set is expected to be valid JSON, which mandates UTF-8 encoding.