PuppetDB 0.9 User's Guide

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

PuppetDB » Overview

Welcome to the beta release of PuppetDB!

PuppetDB is a Puppet data warehouse; it manages storage and retrieval of all platform-generated data. Currently, it stores catalogs and facts; in future releases, it will expand to include more data, like reports.

So far, we've implemented the following features:

- Fact storage
- Full catalog storage
  - Containment edges
  - Dependency edges
  - Catalog metadata
  - Full resource list with all parameters
  - Node-level tags
  - Node-level classes
- REST Fact retrieval
  - all facts for a given node
- REST Resource querying
  - super-set of storeconfigs query API
  - boolean operators
  - can query for resources spanning multiple nodes and types
- Storeconfigs terminus
  - drop-in replacement of stock storeconfigs code
  - export resources
  - collect resources
  - fully asynchronous operation (compiles aren't slowed down)
  - much faster storage, in much less space
- Inventory service API compatibility
  - query nodes by their fact values
  - drop-in replacement for Puppet Dashboard and PE console inventory service

Components

PuppetDB consists of several, cooperating components:

REST-based command processor

PuppetDB uses a CQRS pattern for making changes to its domain objects (facts, catalogs, etc). Instead of simply submitting data to PuppetDB and having it figure out the intent, the intent needs to be explicitly codified as part of the operation. This is known as a "command" (e.g. "replace the current facts for node X").

Commands are processed asynchronously; however, do our best to ensure that once a command has been accepted, it will eventually be executed. Ordering is also preserved. To do this, all incoming commands are placed in a message queue which the command processing subsystem reads from in FIFO order.

Submission of commands is done via HTTP. See the API spec in the navigation sidebar for complete documentation. There is a specific required wire format for commands, and failure to conform to that format will result in an HTTP error.

Storage subsystem

Currently, PuppetDB's data is stored in a relational database. There are two supported databases:

- An embedded HSQLDB. This does not require a separate database service, and is thus trivial to setup. This database is intended for proof-of-concept use; we do not recommend it for long-term production use.
- PostgreSQL

There is no MySQL support, as MySQL lacks support for recursive queries (critical for future graph traversal features).

REST-based retrieval

Read-only requests (resource queries, fact queries, etc.) are done using PuppetDB's REST APIs. Each REST endpoint is documented in the API spec; see the navigation sidebar.

Remote REPL

For debugging purposes, you can open up a remote clojure REPL and use it to modify the behavior of PuppetDB, live.

Vim support would be a welcome addition; please submit patches!

Puppet terminuses

There are a set of Puppet terminuses that acts as a drop-in replacement for stock storeconfigs functionality. By asynchronously storing catalogs in PuppetDB, and by leveraging PuppetDB's fast querying, compilation times are much reduced compared to traditional storeconfigs.

PuppetDB 0.9 » System Requirements

Once installed and configured, PuppetDB will be a critical component of your Puppet deployment, and agent nodes will be unable to request catalogs if it becomes unavailable. In general, it should be run on a robust and reliable server, like any critical service.

Basic Requirements

PuppetDB will run on any *nix system with JDK 1.6 or higher. This includes:

- Recent MacOS X versions using built-in Java support
- Nearly any Linux distribution using its own OpenJDK packages
- Nearly any *nix system running a recent Oracle-provided JDK

Easy Install Requirements

Puppet Labs provides PuppetDB packages for several major Linux distributions; these packages automate the complex process of setting up SSL, and make it easier to keep PuppetDB up to date.

To take advantage of this, you should install PuppetDB on a server running one of the following operating systems:

- Red Hat Enterprise Linux 5 or 6
- Any Linux distro derived from RHEL 5 or 6, including (but not limited to) CentOS, Scientific Linux, and Ascendos
- 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

Puppet Requirements

Any puppet master you wish to connect to PuppetDB must be running Puppet version 2.7.12 or higher. You will also need to install extra components on your puppet master(s) before they can speak to PuppetDB.

After installing PuppetDB, see here to configure a puppet master to use it.

Database Recommendations

Deployments with more than 100 nodes should configure a PostgreSQL database for PuppetDB. Smaller deployments may also wish to use the PostgreSQL backend.

There are two available backends for PuppetDB storage:

- PuppetDB's embedded database
- PostgreSQL

The embedded database works well for small deployments (up to approximately 100 hosts). It requires no additional daemons or setup, and as such is very simple to get started with. It supports all PuppetDB features.

However, there is a cost: the embedded database requires a fair amount of RAM to operate correctly. We'd recommend allocating 1GB to PuppetDB as a starting point. Additionally, the embedded database is somewhat opaque; unlike more off-the-shelf database daemons, there isn't much companion tooling for things like interactive SQL consoles, performance analysis, or backups.

That said, if you have a small installation and enough RAM, then the embedded database will work just fine.

For most "real" use, we recommend running an instance of PostgreSQL. Simply install PostgreSQL using a module from the Puppet Forge or your local package manager, create a new (empty) database for PuppetDB, and verify that you can login via `psql` to this DB you just created. Then just supply PuppetDB with the DB host, port, name, and credentials you've just configured, and we'll take care of the rest!

Memory Recommendations

PuppetDB runs on the JVM, and the maximum amount of memory it is allowed to use is set when the service is started. The optimal value of this maximum will vary depending on the nature of your site.

For Embedded DB Users

If you're using the embedded database, we recommend using 1GB or more (to be on the safe side).

For PostgreSQL users

If you are using an external database, then a decent rule-of-thumb is to allocate 128MB of memory as a base, plus 1MB for each node in your infrastructure. To get a more exact measure of the required RAM, you should start with the rule-of-thumb, then watch the performance console and experiment.

Large-scale Recommendations

For truly large installations, we recommend terminating SSL using Apache or Nginx instead of within PuppetDB itself. This permits much greater flexibility and control over bandwidth and clients. Instructions for configuring this are currently beyond the scope of this manual.

PuppetDB » Known Issues

Significant Bugs

None

If you are aware of any major issues, please file a bug report and we'll get right on it.

Minor Issues

PuppetDB does not restart after running Redhat Package Manager

This is issue #15075. Running RPM will correctly stop PuppetDB if it is enabled, but fails to restart it. You will need to manually restart PuppetDB. Please comment on the bug report if you have any additional information.

Broader Issues

Autorequire relationships are opaque

We don't correctly model dependencies between Puppet resources that have an auto-require relationship. If you have 2 file resources where one is a parent directory of the other, Puppet will automatically create a dependency such that the child requires the parent. The problem is that such a dependency is not reflected in the catalog. As currently represented, the catalog contains no hint that such a relationship ever existed.

PuppetDB 0.9 » Installing PuppetDB

Before installing, review the PuppetDB system requirements.

These package-based install instructions cover the supported versions of enterprise Linux, Debian, Ubuntu, and Fedora. To install on other systems, you should instead see Installing from Source.

Step 1: Install and Configure 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`."

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

- Instructions for Puppet Enterprise
- Instructions for open source Puppet

> 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

For PE Users

On EL systems, run:

```
$ sudo yum install pe-puppetdb
```

On Debian and Ubuntu systems, run:

```
$ sudo apt-get install pe-puppetdb
```

For Open Source Users

On EL and Fedora systems, run:

```
$ sudo yum install puppetdb
```

On Debian and Ubuntu systems, run:

```
$ sudo apt-get install puppetdb
```

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 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, and 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.

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 0.9 » Installing PuppetDB from

Source

If possible, we recommend installing PuppetDB from packages. However, if you are installing PuppetDB on a system without 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 is provided in the root of the source repo, as `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, it has received a valid certificate from your site's Puppet CA, and you installed PuppetDB from source, 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 annoying process, and 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, 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, using 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, and 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 it 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.

PuppetDB 0.9 » Connecting Puppet to PuppetDB

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

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

properly configured puppet master 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

Follow all of the instructions below on your puppet master server(s).

Step 1: Install Plugins

Currently, puppet masters need 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.

For PE Users

- Enable the Puppet Labs repo.
- Install the `pe-puppetdb-terminus` package.
  - On Debian and Ubuntu: run `sudo apt-get install pe-puppetdb-terminus`
  - On EL or Fedora: run `sudo yum install pe-puppetdb-terminus`

For Open Source Users

- Enable the Puppet Labs repo.
- Install the `puppetdb-terminus` package.
  - On Debian and Ubuntu: run `sudo apt-get install puppetdb-terminus`
  - On EL or Fedora: run `sudo yum install puppetdb-terminus`

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:

Edit puppetdb.conf

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

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

- PuppetDB's port for secure traffic defaults to 8081.
- PuppetDB's port for insecure traffic defaults to 8080, but doesn't accept connections by default.

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

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

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: If you previously set the `thin_storeconfigs` or `async_storeconfigs` settings to `true`, you should delete them at this time. The old queuing mechanism will interfere with performance, and thinned catalogs are no longer necessary. Likewise, if you previously used the puppet queue/puppetqd daemon, you should now disable it.

Edit routes.yaml

The routes.yaml file will probably not exist yet. Create it if necessary, and edit it to contain the following:

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

This is necessary for making 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 0.9 » Maintaining and Tuning

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

Deactivate Decommissioned Nodes

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

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

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

Although deactivated nodes will be excluded from storeconfigs queries, their data is still preserved, and a node will be reactivated if a new catalog or facts are received for it.

Redoing SSL setup after changing certificates

If you've recently changed the certificates in use by the PuppetDB server, you'll 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 perform again all the SSL configuration steps outlined in the installation instructions.

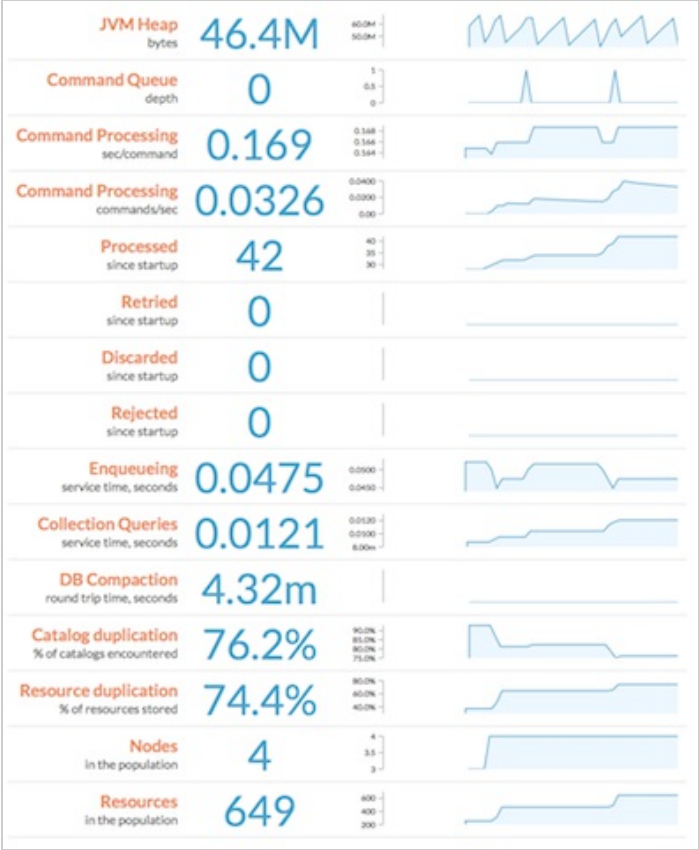
Monitor the Performance Console

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`

PuppetDB uses this page to display a web-based console 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.



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`

> Note: You may need to change PuppetDB's configuration to make the dashboard available, since the default configuration will only allow unauthenticated access to `localhost`. See here to configure unauthenticated HTTP for PuppetDB.

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 troubleshoot visible malfunctions.

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, and everyone's memory needs will be different depending on their 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 guess a ballpark figure, then monitor the performance console 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 console, note the MQ depth. If it is rising and you have CPU cores to spare, increasing the number of threads may help churn through 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.

PuppetDB 0.9 » Configuration

Configuring the Java Heap Size

Unlike most of PuppetDB's settings, the JVM heap size is specified on the command line. If you installed PuppetDB from packages or used the `rake install` installation method, you can change this setting in the init script's configuration file.

The location of this file varies by platform and by the package used to install:

- Redhat-like, with open source package: `/etc/sysconfig/puppetdb`
- Redhat-like, with PE package: `/etc/sysconfig/pe-puppetdb`
- Debian/Ubuntu, with open source package: `/etc/default/puppetdb`
- Debian/Ubuntu, with PE package: `/etc/default/pe-puppetb`

To change the heap size, edit this file and change the `JAVA_ARGS` variable. This setting contains command line flags for the Java executable, and you should use the `-Xmx` flag to specify a max size. For example, to cap PuppetDB at 192MB of memory:

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

To use 1GB of memory:

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

The PuppetDB Configuration File(s)

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

Whenever you change settings in the configuration file, you must restart PuppetDB before the changes will take effect.

Here is an example configuration file:

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

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

[jetty]
port = 8080
```

You can specify either 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, the default is to use the conf.d style, directory-based approach. The default config directory is `/etc/puppetdb/conf.d` (or `/etc/puppetlabs/puppetdb/conf.d` for Puppet Enterprise installations). If you're running from source you may use the "-c" command-line argument to specify your config file or directory.

The available configuration sections and settings are as follows:

`[global]`

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

`vardir`
: 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`
: 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, we default 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.

`[database]`

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

We currently support two configurations: a built-in HSQLDB database, and PostgreSQL. If no database information is supplied, an HSQLDB database at `<vardir>/db` will be used.

UNIVERSAL DB SETTINGS

`gc-interval`
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.

EMBEDDED DB SETTINGS

`classname`, `subprotocol`, and `subname`
These three settings must be configured as follows:

```
classname = org.hsqldb.jdbcDriver
subprotocol = hsqldb
subname = file:/path/to/db;hsqldb.tx=mvcc;sql.syntax_pgs=true
```

Replace `/path/to/db` with a filesystem location in which you'd like to persist the database.

POSTGRESQL DB SETTINGS

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. 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 db 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
```

`classname`, `subprotocol`, and `subname`
These three settings must be configured as follows:

```
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //<host>:<port>/<database>
```

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 indicate how to do this. Be sure to add `ssl=true` to the `subname` parameter.

`username`
The username to use when connecting.

`password`
The password to use when connecting.

`[command-processing]`

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

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

`threads`
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 console.

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

`[jetty]` (HTTP)

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

`host`
The hostname to listen on for unencrypted HTTP traffic. If not supplied, we bind to localhost.
`port`
What port to use for unencrypted HTTP traffic. If not supplied, we won't listen for unencrypted traffic at all.
`ssl-host`
The hostname to listen on for HTTPS. If not supplied, we bind to localhost.
`ssl-port`
What port to use for encrypted HTTPS traffic. If not supplied, we won't listen for encrypted traffic at all.
`keystore`
The path to a Java keystore file containing the key and certificate we should use for HTTPS.
`key-password`
Passphrase to use to unlock the keystore file.
`truststore`
The path to a Java keystore file containing the CA certificate(s) for your puppet infrastructure.
`trust-password`
Passphrase to use to unlock the truststore file.
`certificate-whitelist`
Optional. Path to a file that contains a list of hostnames, one per line. Incoming HTTPS requests will have their certificates validated against this list of hostnames, and only those with an exact, matching entry will be allowed through.

If not supplied, we'll perform standard HTTPS without any additional authorization. We'll still make sure that all HTTPS clients supply valid, verifiable SSL client certificates.

`[repl]` (Remote Runtime Modification)

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 0.9 » Using PuppetDB

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

If you wish to build applications on PuppetDB, see the navigation sidebar for 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>
```

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 to be used 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 — inventory information will be coming from PuppetDB as soon as the puppet master is connected to it.

PuppetDB 0.9 » 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, and 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, you should consider running PuppetDB directly from source.

PuppetDB 0.9 » Spec » Commands

Commands are the mechanism by which changes are made to 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 identifier for the desired 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 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, and will be used for verification.□

If a command has been successfully submitted, the submitter will receive the following response:

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

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 around when that command will be processed, or 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.

PuppetDB 0.9 » Spec » Querying Nodes

Querying nodes is accomplished 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 is a similar format to resource queries.

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

These 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.

This 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 matching the predicates, sorted in ascending order:

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

PuppetDB 0.9 » Spec » Querying Facts

Querying facts is accomplished by making an HTTP request to the `/facts` REST endpoint.

Query format

Facts are queried by making a request to a URL of 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 indicaates 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.

PuppetDB 0.9 » Spec » Querying Resources

Querying resources is accomplished by making 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 is described by the following grammar:

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

`field` may be any of:

`tag`
: a 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 <resource_param>]`
: 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 the resource was declared in

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

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

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

The conditional type behaviours 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 match operator behaviours are defined:

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

Response format

An array of zero or more resource objects, with each object having 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>,
                 ...}}
```

PuppetDB 0.9 » Spec » Querying Status

Query Format

Node status can be queried by making a GET request to `/status/nodes/<node>`, accepting 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.

PuppetDB 0.9 » 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`.
- A `GET` request is used.
- There is an `Accept` header containing `application/json`.

Response format

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 returned in the list of available metrics specified above.
- A `GET` request is used.
- There is an `Accept` header containing `application/json`.

Response format

A JSON Object mapping strings to (strings/numbers/booleans).

Example

You can use `curl` to grab metrics like so:

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

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 catalog replacement

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 that 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 around 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, stats around how long messages remain in the queue, etc.

PuppetDB 0.9 » Spec » Catalog Wire Format

PuppetDB receives a new, specifically modified catalog wire format from puppet masters.

Previous Wire Format Shortcomings

There are 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) can handle this data.

New Catalog Interchange format

A catalog is serialized as JSON (which implies UTF-8 encoding). Unless otherwise noted, null is not allowed anywhere in the catalog.

```
{"metadata": {
    "type": "catalog",
    "version": 1
    }
 "data": {
    "name": <string>,
```

```
    "version": <string>,
    "classes":
        [<string>, <string>, ...],
    "tags":
        [<string>, <string>, ...],
    "edges":
        [<edge>, <edge>, ...],
    "resources":
        [<resource>, <resources>, ...]
    }
}
```

All keys are mandatory, although values that are lists may be empty lists.

`"name"` is the certname the catalog is associated with.

`"version"` is an arbitrary tag that uniquely identifies this catalog across time for a single node.□

Encoding

The entire catalog is expected to be valid JSON, which requires strict UTF-8 encoding.

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:

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

All keys for `edge` are required.

Data type: `<resource-spec>`

Synonym: `<resource-hash>`. A JSON Object of the following form:

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

For every resource-spec listed in the catalog, there must be a corresponding resource in the "resources" list with matching type and title attributes. Among other things, this means that edges refer to resources by their resource-spec, and not by an alias or implicit namevar.

Data type: `<relationship>`

One of the following strings:

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

This mirrors Puppet's `->` construct in terms of defining the source and the target of an edge.

Data type: `<resource>`

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

All keys for `resource` are required.

The `"aliases"` list must include all aliases for a resource beyond the title itself. This includes names in explicit `"alias"` parameters, or implied namevars of the type itself.

Differences from Current Wire Format

1. The format is fully documented here.
2. Information that previously had to be deduced by the puppetmaster 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 0.9 » 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 value MUST be strings.

Encoding

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