PuppetDB 1.3 User's Guide

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

PuppetDB 1.3 » 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).
- If you'd like to migrate existing exported resources from your ActiveRecord storeconfigs database, please see the documentation on Migrating Data.
- Choose your installation method:
  - easy install using the PuppetDB puppet module on our recommended platforms
  - install from packages on our recommended platforms
  - advanced install on any other *nix.

Version Note

This manual covers the 1.3.x series of PuppetDB releases, which is backwards-compatible with the 1.1.x series but adds several new features. See the release notes for information on all changes since the final 1.0.x release.

What Data?

In version 1.3, PuppetDB stores:

- The most recent facts from every node
- The most recent catalog for every node
- Optionally, seven days (configurable) of event reports 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 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 and a Puppet module 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 12.10, 12.04 LTS, 10.04 LTS, 8.04 LTS, 11.10
- Fedora 16, 17 or 18

See here for instructions for installing via the PuppetDB puppet module.

See here for instructions for installing from OS packages.

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.

NOTE ABOUT PUPPET ENTERPRISE

- Due to packaging changes related to future integration of PuppetDB with Puppet Enterprise, PuppetDB 1.3 packages are not available for use with Puppet Enterprise 2.8 or below. Users of Puppet Enterprise 2.8 or below can use PuppetDB 1.1, and should visit the PuppetDB 1.1 documentation. PuppetDB will be included by default in the next major release of Puppet Enterprise.

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.

Frequently Asked Questions

Can I migrate my data from ActiveRecord storeconfigs or from an existing PuppetDB to a new instance?

Yes. At this time, you can only migrate exported resources from ActiveRecord, and you can migrate catalogs from an existing PuppetDB. For more information, see Migrating Data for more information.

PuppetDB is complaining about a truststore or keystore file. What do I do?

There are several related causes for this, but it boils down to PuppetDB being unable to read your truststore.jks or keystore.jks file. The former file contains the certificate for your certificate authority, and is what PuppetDB uses to authenticate clients. The latter contains the key and certificate that PuppetDB uses to identify itself to clients.

The short answer: you can often fix these problems by reinitializing the keystore and truststore, by running `/usr/sbin/puppetdb-ssl-setup`. Note that this script must be run after a certificate is generated for the puppet agent (that is: after the agent has run once and had its certificate request signed). A common problem is installing PuppetDB before the Puppet agent has run, and this script will solve that problem, and many others.

The long answer: if the `puppetdb-ssl-setup script` doesn't solve your problem or if you're curious what's going on under the covers, you can manage this configuration by hand. The locations of the truststore and keystore files are set with the `keystore` and `truststore` options in the config file. There should also be settings for `key-password` and `trust-password`. Make sure the keystore.jks and truststore.jks files are where the config says they should be, and that they're readable by the user PuppetDB runs as (puppetdb for an open source installation, pe-puppetdb for a Puppet Enterprise installation). Additionally, you can verify that the password is correct using `keytool -keystore /path/to/keystore.jks` and and entering the `key-password`. Similarly, you can use `keytool -keystore /path/to/truststore.jks` to verify the truststore.

The PuppetDB dashboard gives me a weird SSL error when I visit it. What gives?

There are two common error cases with the dashboard:

- You're trying to talk over plaintext (8080) and PuppetDB's not listening

By default, PuppetDB only listens for plaintext connections on localhost, for security reasons. In order to talk to it this way, you'll need to either forward the plaintext port or change the interface PuppetDB binds on to one that is accessible from the outside world. In the latter case, you'll want to use some other means to secure PuppetDB (for instance, by restricting which hosts are allowed to talk to PuppetDB through a firewall).

- You're trying to talk over SSL and nobody trusts anybody else

Because PuppetDB uses the certificate authority of your Puppet infrastructure, and a certificate signed by it, PuppetDB doesn't trust your browser, and your browser doesn't trust PuppetDB. In this case, you'll need to give your browser a certificate signed by your Puppet CA. Support for client certificates is varied between browsers, so it's preferred to connect over plaintext, as outlined above.

Does PuppetDB support Puppet apply?

Partially. Use with Puppet apply requires some special configuration, and due to limitations in Puppet, inventory service functionality isn't fully supported. Catalog storage and collection queries are completely functional, though. You can find information about configuring Puppet apply to work with PuppetDB in the installation guide for your version of PuppetDB.

Either of these issues can also be solved through clever and judicious use of proxies, although the details of that are left as an exercise to the reader.

Why is PuppetDB written in Java?

Actually, PuppetDB isn't written in Java at all! It's written in a language called Clojure, which is a dialect of Lisp that runs on the Java Virtual Machine. Several other languages were prototyped, including Ruby and JRuby, but they lacked the necessary performance. We chose to use a JVM language because of its excellent libraries and high performance. Of the available JVM languages, we used Clojure because of its expressiveness, performance, and previous experience with the language on our team.

Which versions of Java are supported?

The officially supported versions are OpenJDK and Oracle JDK, versions 1.6 and 1.7. Other versions may work and issues will be addressed on a best effort basis, but support is not guaranteed.

Which databases are supported?

PostgreSQL is the recommended database for production use. PuppetDB also ships with an embedded HyperSQL database which is suitable for very small or proof of concept deployments. As with our choice of language, we prototyped several databases before settling on PostgreSQL. These included Neo4j, Riak, and MySQL with ActiveRecord in Ruby. We have no plans to support any other databases, including MySQL, which lacks important features such as array columns and recursive queries.

I may have a corrupt KahaDB store. What does this mean, what causes it and how can I recover?

If PuppetDB throws an exception while the application starts or while receiving a command it may be due to KahaDB corruption. The exception generally has some mention of the KahaDB libraries (org.apache.activemq.store.kahadb), for example:

```
java.io.EOFException
    at java.io.RandomAccessFile.readInt(RandomAccessFile.java:776)
    at
org.apache.activemq.store.kahadb.disk.journal.DataFileAccessor.readRecord(DataFile
```

You should consult the Troubleshooting guide for Kahadb for details on how to rememdy this.

PuppetDB 1.3 » Release Notes

1.3.2

PuppetDB 1.3.2 is a bugfix release. Many thanks to the following people who contributed patches to this release:

- Chris Price

Bug fixes:

- Size of column `puppet_version` in the database schema is insufficient

  There is a field in the database that is used to store a string representation of the puppet version along with each report. Previously, this column could contain a maximum of 40 characters, but for certain builds of Puppet Enterprise, the version string could be longer than that. This change simply increases the maximum length of the column.

1.3.1

PuppetDB 1.3.1 is a bugfix release. Many thanks to the following people who contributed patches to this release:

- Chris Price
- Deepak Giridharagopal
- Ken Barber
- Matthaus Owens
- Nick Fagerlund

Bug fixes:

- (#19884) Intermittent SSL errors in Puppet master / PuppetDB communication

  There is a bug in OpenJDK 7 (starting in 1.7 update 6) whereby SSL communication using Diffie-Hellman ciphers will error out a small percentage of the time. In 1.3.1, we've made the list of SSL ciphers that will be considered during SSL handshake configurable. In addition, if you're using an affected version of the JDK and you don't specify a legal list of ciphers, we'll automatically default to a list that does not include the Diffie-Hellman variants. When this issue is fixed in the JDK, we'll update the code to re-enable them on known good versions.

- (#20563) Out of Memory error on PuppetDB export

  Because the `puppetdb-export` tool used multiple threads to retrieve data from PuppetDB and a single thread to write the data to the export file, it was possible in certain hardware configurations to exhaust all of the memory available to the JVM. We've moved this back to a single-threaded implementation for now, which may result in a minor performance decrease for exports, but will prevent the possibility of hitting an OOM error.

- Don't check for newer versions in the PE-PuppetDB dashboard

  When running PuppetDB as part of a Puppet Enterprise installation, the PuppetDB package should not be upgraded independently of Puppet Enterprise. Therefore, the notification message that would appear in the PuppetDB dashboard indicating that a newer version is available has been removed for PE environments.

1.3.0

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

- Branan Purvine-Riley
- Chris Price
- Deepak Giridharagopal
- Ken Barber
- Matthaus Owens
- Moses Mendoza
- Nick Fagerlund
- Nick Lewis

Notable features:

- Report queries

  The query endpoint `experimental/event` has been augmented to support a much more interesting set of queries against report data. You can now query for events by status (e.g. `success`, `failed`, `noop`), timestamp ranges, resource types/titles/property name, etc. This should make the report storage feature of PuppetDB much more valuable!

- Import/export of PuppetDB reports

  PuppetDB 1.2 added the command-line tools `puppetdb-export` and `puppetdb-import`, which are useful for migrating catalog data between PuppetDB databases or instances. In PuppetDB 1.3, these tools now support importing and exporting report data in addition to catalog data.

Bug fixes:

- `puppetdb-ssl-setup` is now smarter about not overwriting keystore settings in `jetty.ini` during upgrades
- Add database index to `status` field for events to improve query performance
- Fix `telnet` protocol support for embedded nrepl
- Upgrade to newer version of nrepl
- Improvements to developer experience (remove dependency on `rake` for building/running clojure code)

1.2.0

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

- Chris Price
- Deepak Giridharagopal

- Erik Dalén
- Jordi Boggiano
- Ken Barber
- Matthaus Owens
- Michael Hall
- Moses Mendoza
- Nick Fagerlund
- Nick Lewis

Notable features:

- Automatic node purging

  This is the first feature which allows data in PuppetDB to be deleted. The new `node-purge-ttl` setting specifies a period of time to keep deactivated nodes before deleting them. This can be used with the `puppet node deactivate` command or the automatic node deactivation `node-ttl` setting. This will also delete all facts, catalogs and reports for the purged nodes. As always, if new data is received for a deactivated node, the node will be reactivated, and thus exempt from purging until it is deactivated again. The `node-purge-ttl` setting defaults to 0, which disables purging.

- Import/export of PuppetDB data

  Two new commands have been added, `puppetdb-export` and `puppetdb-import`. These will respectively export and import the entire collection of catalogs in your PuppetDB database. This can be useful for migrating from HSQL to PostgreSQL, for instance.

  There is also a new Puppet subcommand, `puppet storeconfigs export`. This command will generate a similar export data from the ActiveRecord storeconfigs database. Specifically, this includes only exported resources, and is useful when first migrating to PuppetDB, in order to prevent failures due to temporarily missing exported resources.

- Automatic dead-letter office compression

  When commands fail irrecoverably or over a long period of time, they are written to disk in what is called the dead-letter office (or DLO). Until now, this directory had no automatic maintenance, and could rapidly grow in size. Now there is a `dlo-compression-threshold` setting, which defaults to 1 day, after which commands in the DLO will be compressed. There are also now metrics collected about DLO usage, several of which (size, number of messages, compression time) are visible from the PuppetDB dashboard.

- Package availability changes

  Packages are now provided for Fedora 18, but are no longer provided for Ubuntu 11.04 Natty Narwhal, which is end-of-life. Due to work being done to integrate PuppetDB with Puppet Enterprise, new pe-puppetdb packages are not available.

Bug fixes:

- KahaDB journal corruption workaround

  If the KahaDB journal, used by ActiveMQ (in turn used for asynchronous message processing), becomes corrupted, PuppetDB would fail to start. However, if the embedded ActiveMQ broker is

restarted, it will cleanup the corruption itself. Now, PuppetDB will recover from such a failure and restart the broker automatically.

- Terminus files conflict between puppetdb-terminus and puppet

  There was a conflict between these two packages over ownership of certain directories which could cause the puppetdb-terminus package to fail to install in some cases. This has been resolved.

1.1.1

PuppetDB 1.1.1 is a bugfix release. It contains the following fixes:

- (#18934) Dashboard Inventory Service returns 404

  Version 1.1.0 of the PuppetDB terminus package contained a faulty URL for retrieving fact data for the inventory service. This issue is fixed and we've added better testing to ensure that this doesn't break again in the future.

- (#18879) PuppetDB terminus 1.0.5 is incompatible with PuppetDB 1.1.0

  Version 1.1.0 of the PuppetDB server package contained some API changes that were not entirely backward-compatible with version 1.0.5 of the PuppetDB terminus; this caused failures for some users if they upgraded the server to 1.1.0 without simultaneously upgrading the terminus package. Version 1.1.1 of the server is backward-compatible with terminus 1.0.5, allowing an easier upgrade path for 1.0.x users.

1.1.0

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

- Chris Price
- Deepak Giridharagopal
- Jeff Blaine
- Ken Barber
- Kushal Pisavadia
- Matthaus Litteken
- Michael Stahnke
- Moses Mendoza
- Nick Lewis
- Pierre-Yves Ritschard

Notable features:

- Enhanced query API

  A substantially improved version 2 of the HTTP query API has been added. This is located under the /v2 route. Detailed documentation on all the available routes and query language can be found in the API documentation, but here are a few of the noteworthy improvements:

  - Query based on regular expressions

    Regular expressions are now supported against most fields when querying against resources, facts, and nodes, using the ~ operator. This makes it easy to, for instance, find all IP addresses

for a node, or apply a query to some set of nodes.

- More node information

  Queries against the /v2/nodes endpoint now return objects, rather than simply a list of node names. These are effectively the same as what was previously returned by the /status endpoint, containing the node name, its deactivation time, as well as the timestamps of its latest catalog, facts, and report.

- Full fact query

  The /v2/facts endpoint supports the same type of query language available when querying resources, where previously it could only be used to retrieve the set of facts for a given node. This makes it easy to find the value of some fact for all nodes, or to do more complex queries.

- Subqueries

  Queries can now contain subqueries through the `select-resources` and `select-facts` operators. These operators perform queries equivalent to using the /v2/resources and /v2/facts routes, respectively. The information returned from them can then be correlated, to perform complex queries such as “fetch the IP address of all nodes with `Class[apache]`”, or “fetch the `operatingsystemrelease` of all Debian nodes”. These operators can also be nested and correlated on any field, to answer virtually any question in a single query.

- Friendlier, RESTful query routes

  In addition to the standard query language, there are also now more friendly, “RESTful” query routes. For instance, `/v2/nodes/foo.example.com` will return information about the node foo.example.com. Similarly, `/v2/facts/operatingsystem` will return the `operatingsystem` of every node, or `/v2/nodes/foo.example.com/operatingsystem` can be used to just find the `operatingsystem` of foo.example.com.

  The same sort of routes are available for resources as well. `/v2/resources/User` will return every User resource, `/v2/resources/User/joe` will return every instance of the `User[joe]` resource, and `/v2/nodes/foo.example.com/Package` will return every Package resource on foo.example.com. These routes can also have a query parameter supplied, to further query against their results, as with the standard query API.

- Improved catalog storage performance

  Some improvements have been made to the way catalog hashes are computed for deduplication, resulting in somewhat faster catalog storage, and a significant decrease in the amount of time taken to store the first catalog received after startup.

- Experimental report submission and storage

  The ‘puppetdb’ report processor is now available, which can be used (alongside any other reports) to submit reports to PuppetDB for storage. This feature is considered experimental, which means the query API may change significantly in the future. The ability to query reports is currently limited and experimental, meaning it is accessed via /experimental/reports rather than /v2/reports. Currently it is possible to get a list of reports for a node, and to retrieve the contents of a single report. More advanced querying (and integration with other query endpoints) will come in a future release.

  Unlike catalogs, reports are retained for a fixed time period (defaulting to 7 days), rather than

only the most recent report being stored. This means more data is available than just the latest, but also prevents the database from growing unbounded. See the documentation for information on how to configure the storage duration.

- Tweakable settings for database connection and ActiveMQ storage

  It is now possible to set the timeout for an idle database connection to be terminated, as well as the keep alive interval for the connection, through the `conn-max-age` and `conn-keep-alive` settings.

  The settings `store-usage` and `temp-usage` can be used to set the amount of disk space (in MB) for ActiveMQ to use for permanent and temporary message storage. The main use for these settings is to lower the usage from the default of 100GB and 50GB respectively, as ActiveMQ will issue a warning if that amount of space is not available.

Behavior changes:

- Messages received after a node is deactivated will be processed

  Previously, commands which were initially received before a node was deactivated, but not processed until after (for instance, because the first attempt to process the command failed, and the node was deactivated before the command was retried) were ignored and the node was left deactivated. For example, if a new catalog were submitted, but couldn't be processed because the database was temporarily down, and the node was deactivated before the catalog was retried, the catalog would be dropped. Now the catalog will be stored, though the node will stay deactivated. Commands received after a node is deactivated will continue to reactivate the node as before.

1.0.5

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

- Chris Price
- Deepak Giridharagopal

Fixes:

- Drop a large, unused index on catalog_resources(tags)

  This index was superseded by a GIN index on the same column, but the previous index was kept around by mistake. This should result in a space savings of 10–20%, as well as a possible very minor improvement in catalog insert performance.

1.0.4

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

- Chris Price

Fixes:

- (#16554) Fix postgres query for numeric comparisons

  This commit changes the regex that we are using for numeric comparisons in postgres to a format that is compatible with both 8.4 and 9.1.

1.0.3

NOTE: This version was not officially released, as additional fixes came in between the time we tagged this and the time we were going to publish release artifacts.

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

- Deepak Giridharagopal
- Nick Lewis
- Chris Price

Fixes:

- (#17216) Fix problems with UTF-8 transcoding

  Certain 5 and 6 byte sequences were being incorrectly transcoded to UTF-8 on Ruby 1.8.x systems. We now do two separate passes, one with iconv and one with our hand-rolled transcoding algorithms. Better safe than sorry!

- (#17498) Pretty-print JSON HTTP responses

  We now output more nicely-formatted JSON when using the PuppetDB HTTP API.

- (#17397) DB pool setup fails with numeric username or password

  This bug happens during construction of the DB connection pool. If the username or password is numeric, when parsing the configuration file they're turned into numbers. When we go to actually create the pool, we get an error because we're passing in numbers when strings are expected.

- (#17524) Better logging and response handling for version checks

  Errors when using the `version` endpoint are now caught, logged at a more appropriate log level, and a reasonable HTTP response code is returned to callers.

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 1.3 » 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 1.3 » 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.

Erik 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 1.3 » Migrating Data

Migrating from ActiveRecord storeconfigs

If you're using exported resources with ActiveRecord storeconfigs, you may want to migrate your existing data to PuppetDB before connecting the master to it. This will ensure that whatever resources were being collected by the agents will still be collected, and no incorrect configuration will be applied.

The existing ActiveRecord data can be exported using the `puppet storeconfigs export` command, which will produce a tarball that can be consumed by PuppetDB. Because this command is intended only to stop nodes from failing until they have check into PuppetDB, it will only include exported resources, excluding edges and facts.

NOTE: in order for this to work properly, you need to make sure you've run this command and generated the export tarball prior to configuring your master for PuppetDB.

Once you've run this command and generated an export tarball, you should follow the instructions below to import the tarball into your PuppetDB database.

Exporting data from an existing PuppetDB database

If you've been trying out PuppetDB using the embedded database and are ready to move to a production environment backed by PostgreSQL, or if you'd simply like to move your data from one PostgreSQL database to another one, you can use the `puppetdb-export` command (which is available in your `/usr/sbin` directory for versions of PuppetDB >= 1.2). All you'll need to do is run a command like this:

```
$ sudo puppetdb-export --outfile ./my-puppetdb-export.tar.gz
```

This command is intended to be run on the PuppetDB server, and assumes that PuppetDB is accepting plain-text HTTP connections on `localhost` port `8080` (which is PuppetDB's default

configuration). If you've modified your PuppetDB HTTP configuration, you can specify a different hostname and port on the command line. For more info, run:

```
$ sudo puppetdb-export --help
```

While its not required it is recommended you run this tooling while there is no activity on your existing PuppetDB to ensure your data snapshot is consistent. Also the tool can put load on your production system, so you should plan for this before running it.

The generated tarball will contain a backup of all of your current catalog data (including exported resources) and all you report data. At this time fact data exporting is not supported.

Exporting data from a version of PuppetDB prior to 1.2

The `puppetdb-export` and `puppetdb-import` tools were added to PuppetDB in version 1.2. If you need to export data from an older version of PuppetDB, the easiest way to do so is to upgrade your existing PuppetDB to at least version 1.2 and then use the `puppetdb-export` tool.

Importing data to a new PuppetDB database

Once you have an export tarball and a new PuppetDB server up and running that you would like to import your data into, use the `puppetdb-import` command to do so. (This command is available in your `/usr/sbin` directory in versions of PuppetDB >= 1.2.) The syntax will look something like this:

```
$ sudo puppetdb-import --infile ./my-puppetdb-export.tar.gz
```

This command is intended to be run on the new PuppetDB server, and assumes that PuppetDB is accepting plain-text HTTP connections on `localhost` port `8080` (which is PuppetDB's default configuration). If you've modified your PuppetDB HTTP configuration, you can specify a different hostname and port on the command line. For more info, run:

```
$ sudo puppetdb-import --help
```

PuppetDB 1.3 » Installing PuppetDB Via Module

You can install and configure all of PuppetDB's components and prerequisites (including PuppetDB itself, PostgreSQL, firewall rules on RedHat-like systems, and the terminus plugins for your Puppet master) using the PuppetDB module from the Puppet Forge.

- If you are already familiar with Puppet and have a working Puppet deployment, this is the easiest method for installing PuppetDB. In this guide, we expect that you already know how to assign Puppet classes to nodes.
- If you are just getting started with Puppet, you should probably follow the Installing PuppetDB From Packages guide instead.

> Note:

> If you'd like to migrate existing exported resources from your ActiveRecord storeconfigs database, please see the documentation on Migrating Data.

Step 1: Enable the Puppet Labs Package Repository

If you haven't already, you will need to do one of the following:

- Enable the Puppet Labs package repository on your PuppetDB server and puppet master server.
- Grab the PuppetDB and terminus plugin packages, and import them into your site's local package repos.

Step 2: Assign Classes to Nodes

Using the normal methods for your site, assign the PuppetDB module's classes to your servers. You have three main options for deploying PuppetDB:

- If you are installing PuppetDB on the same server as your puppet master, assign the `puppetdb` and `puppetdb::master::config` classes to it.
- If you want to run PuppetDB on its own server with a local PostgreSQL instance, assign the `puppetdb` class to it, and assign the `puppetdb::master::config` class to your puppet master. Make sure to set the class parameters as necessary.
- If you want PuppetDB and PostgreSQL to each run on their own servers, assign the `puppetdb::server` class and the `puppetdb::database::postgresql` classes to different servers, and the `puppetdb::master::config` class to your puppet master. Make sure to set the class parameters as necessary.

Note: by default the module sets up the PuppetDB dashboard to be accessible only via `localhost`. If you'd like to allow access to the PuppetDB dashboard via an external network interface, you should set the `listen_address` parameter on either of the `puppetdb` or `puppetdb::server` classes. e.g.:

```
class { 'puppetdb':
    listen_address => 'example.foo.com'
}
```

These classes automatically configure most aspects of PuppetDB. If you need to set additional settings (to change the `node_ttl`, for example), see the "Playing Nice With the PuppetDB Module" section of the "Configuring" page.

For full details on how to use the module, see the PuppetDB module documentation on Puppet Forge. The module also includes some sample manifests in the `tests` directory that demonstrate its basic usage.

PuppetDB 1.3 » Installing PuppetDB From Packages

This page describes how to manually install and configure PuppetDB from the official packages.

- If you are just getting started with Puppet and don't yet know how to assign Puppet classes to

nodes, this is the guide for you.

- If you are already familiar with Puppet and have a working Puppet deployment, we recommend that you use the puppetlabs-puppetdb module instead. See the "Install via Module" page for more details.

Additionally, these instructions may be useful for understanding the various moving parts, or in cases where you must create your own PuppetDB module.

Notes:

- If you'd like to migrate existing exported resources from your ActiveRecord storeconfigs database, please see the documentation on Migrating Data.
- 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.

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.

Step 3: Install PuppetDB

Use Puppet to install PuppetDB:

```
$ 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.

```
$ 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`.
- 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` 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 1.3 » Installing PuppetDB from Source

This page describes how to install PuppetDB from an archive of the source code, or alternately how to run it directly from source without installing.

If possible, we recommend installing PuppetDB with the puppetlabs-puppetdb module or from

packages; either approach will be easier. However, if you are testing a new version, developing PuppetDB, or installing it on a system not supported with official packages, you will need to install it from source.

> Note:
>
> If you'd like to migrate existing exported resources from your ActiveRecord storeconfigs database, please see the documentation on Migrating Data.

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 1.3 » 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.

Migrating existing data

If you are not planning to change your underlying PuppetDB database configuration prior to upgrading, you don't need to worry about migrating your existing data; PuppetDB will handle this automatically. However, if you do plan to switch to a different database, you should export your existing data prior to changing your database configuration. For more information about the PuppetDB import and export tools, please see the documentation on Migrating Data.

Upgrading with the PuppetDB Module

If you installed PuppetDB with the module, you only need to do the following to upgrade:

1. If you imported the official packages into your local package repositories, import the new versions of the PuppetDB and terminus plugin packages.
2. Change the value of the `puppetdb_version` parameter for the `puppetdb` or `puppetdb::server` and `puppetdb::master::config` classes, unless it was set to `latest`.
3. If you are doing a large version jump, trigger a Puppet run on the PuppetDB server before the puppet master server has a chance to do a Puppet run. (It's possible for a new version of the terminus plugins to use API commands unsupported by old PuppetDB versions, which would cause Puppet failures until PuppetDB was upgraded, but this should be very rare.)

Manually Upgrading PuppetDB

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.

```
$ 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:

```
$ 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 1.3 » 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

> Note: if you've installed PuppetDB using the PuppetDB puppet module, then the `puppetdb::master::config` class is taking care of all of this for you.

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.

On Platforms With Packages

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 ext/master/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.

ENABLING EXPERIMENTAL REPORT STORAGE

Version 1.1 of PuppetDB includes experimental support for storing Puppet reports. This feature can be enabled by simply adding the `puppetdb` report processor in your `puppet.conf` file. If you don't already have a `reports` setting in your `puppet.conf` file, you'll probably want to add a line like this:

```
reports = store,puppetdb
```

This will keep Puppet's default behavior of storing the reports to disk as YAML, while also sending the reports to PuppetDB.

You can configure how long PuppetDB stores these reports, and you can do some very basic querying. For more information, see:

- The experimental `event` query endpoint
- The experimental `report` query endpoint
- The experimental `store report` command
- The experimental report wire format

More information about Puppet report processors in general can be found here.

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.

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` 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 1.3 » 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 `puppetdb-terminus` package is installed:

```
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 1.3 » 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-puppetdb`

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 JMX Access

While all JMX metrics are exposed using the `/metrics` namespace, you can also expose direct JMX access using standard JVM means as documented here. This can be done using the `JAVA_ARGS` init script setting, similar to configuring the heap size.

For example, adding the following JVM options will open up a JMX socket on port 1099:

```
JAVA_ARGS="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=1099"
```

The log4j Logging Config File

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. All of the sections and settings described below belong in the PuppetDB config file(s).

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
```

Playing Nice With the PuppetDB Module

If you installed PuppetDB with the puppetlabs-puppetdb module, the config file(s) will be managed by Puppet. However, since the module manages these files on a per-setting basis, you can still configure additional settings that the module doesn't set.

To do this, you should create a new class (something like `site::puppetdb::server::extra`), declare any number of `ini_setting` resources as shown below, set the class to refresh the

`puppetdb::server` class, and assign it to your PuppetDB server.

```
    # Site-specific PuppetDB settings. Declare this class on any node that gets
the puppetdb::server class.
    class site::puppetdb::server::extra {

      # Restart the PuppetDB service if settings change
      Class[site::puppetdb::server::extra] ~> Class[puppetdb::server]

      # Get PuppetDB confdir
      include puppetdb::params
      $confdir = $puppetdb::params::confdir

      # Set resource defaults assuming we're only doing [database] settings
      Ini_setting {
        path => "${confdir}/database.ini",
        ensure => present,
        section => 'database',
        require => Class['puppetdb::server::validate_db'],
      }

      ini_setting {'puppetdb_node_ttl':
        setting => 'node_ttl',
        value => '5d',
      }

      ini_setting {'puppetdb_report_ttl':
        setting => 'report_ttl',
        value => '30d',
      }

    }
```

`[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.

`event-query-limit`

The maximum number of legal results that a resource event 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.

`update-server`

The URL to query when checking for newer versions; defaults to `http://updates.puppetlabs.com/check-for-updates`. Overriding this setting may be useful if your PuppetDB server is firewalled and can't make external HTTP requests, in which case you could configure a proxy server to send requests to the `updates.puppetlabs.com` URL and override this setting to point to your proxy server.

`[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.

USING SSL WITH POSTGRESQL

It's possible to use SSL to protect connections to the database. There are several extra steps and considerations when doing so; see the PostgreSQL SSL setup page for complete details.

The main difference in the config file is that you must 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`

Auto-deactivate nodes that haven't seen any activity (no new catalogs, facts, etc) in the specified amount of time. You may specify the time as a string using any of the following suffixes:

```
`d`   - days
`h`   - hours
`m`   - minutes
`s`   - seconds
```

```
  `ms` - milliseconds
```

So, e.g., a value of `30d` would set the time-to-live to 30 days, and a value of `48h` would set the time-to-live to 48 hours.

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.

`node-purge-ttl`

Automatically delete nodes that have been deactivated for the specified amount of time. This will also delete all facts, catalogs and reports for the node. This ttl may be specified the same way as `node-ttl` above.

If unset, auto-deletion of nodes is disabled.

`report-ttl`

Automatically delete reports that are older than the specified amount of time. You may specify the time as a string using any of the suffixes described in the `node-ttl` section above.

Outdated reports will be deleted during the database garbage collection, which runs every `gc-interval` minutes.

If unset, the default value is 14 days.

`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.

`dlo-compression-threshold`

This setting specifies the maximum duration to keep messages in the dead-letter office before archiving them. This process will check for compressible messages on startup and after every `gc-interval`, but will only perform the archive once per `dlo-compression-threshold`. The same format can be used as for the `node-ttl` setting above. If set to 0 seconds, this behavior will be disabled. The default value is 1 day.

`[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.

`cipher-suites`

Optional. A comma-separated list of cryptographic ciphers to allow for incoming SSL connections. Valid names are listed in the official JDK cryptographic providers documentation; you'll need to use the all-caps cipher suite name.

If not supplied, PuppetDB uses the default cipher suites for your local system on JDK versions older than 1.7.0u6. On newer JDK versions, PuppetDB will use only non-DHE cipher suites.

`[repl]` Settings

The `[repl]` section configures remote runtime modification. For more detailed info, see Debugging with the Remote REPL.

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` or `telnet`.

The telnet repl type opens up a socket you can connect to via telnet. The interface is pretty low-level and raw (no completion or command history), but it is nonetheless usable on just about any system without the need of any external tools other than telnet itself.

The nrepl repl type opens up a socket you can connect to via any nrepl-protocol client, such as via Leiningen using `lein repl :connect localhost:8082` or via Emacs (via `M-x nrepl`), Vim, or integration with other editors like Netbeans or Eclipse. This is much more user-friendly than telnet.

The swank type allows emacs' clojure-mode to connect directly to a running PuppetDB instance by using `M-x slime-connect`. This is not recommended, as the upstream Swank project has been deprecated in favor of nrepl.

`port`

The port to use for the REPL.

PuppetDB 1.3 » Configuration » Using SSL with PostgreSQL

Talking to PostgreSQL using SSL/TLS

If you want SSL/TLS-secured connectivity between PuppetDB and PostgreSQL, you can configure it by following the instructions below.

When configuring SSL you need to decide whether you will:

- Use a self-signed certificate on the DB server (for example, you can use the Puppet CA for this)
- Use a publicly signed certificate on the DB server

Both methodologies are valid, but while self-signed certificates are by far more common in the real world, these types of configurations must be setup with some care.

If you wish to use self-signed certificates then you have a few options:

- Place your private CA in a Java Keystore and tell Java to use that keystore for its system keystore
- Disable SSL verification

Each option has different impact on your configuration, so we will try to go into more detail below.

Before beginning, take a look at the documentation Secure TCP/IP Connections with SSL as this explains how to configure SSL on the server side in detail.

Note: At this point the documentation below only covers server-based SSL, client certificate support is not documented.

Using Puppet Certificates with the Java Keystore

In this case we use the Puppet certificates to secure your PostgreSQL server. This has the following benefits:

- Since your using PuppetDB we can presume that your are using Puppet on each server, this means you can re-use the local Puppet Agent certificate for PostgreSQL.
- Since your local agents certificate must be signed for Puppet to work, most people already have a process for getting these signed which reduces the steps required compared to some other self-signed workflow.
- We already recommend this methodology for securing the HTTPS interface for PuppetDB, so less effort again.

To begin, configure your PostgreSQL server to use the hosts Puppet server certificate and key. The location of these files can be found by using the following commands:

```
# Certificate
puppet config print hostcert
# Key
puppet config print hostprivkey
```

These files will need to be copied to the relevant directories as specified by the PostgreSQL configuration items `ssl_cert_file` and `ssl_key_file` as explained in detail by the Secure TCP/IP Connections with SSL instructions.

You will also need to make sure the setting `ssl` is set to `on` in your `postgresql.conf`. Once this has been done, restart PostgreSQL.

Now continue by creating a truststore as specified in the setup instructions for PuppetDB. If you have installed PuppetDB using a package or you have already used the tool `puppetdb-ssl-setup`, this will most likely already exist in `/etc/puppetdb/ssl`.

You will then need to tell Java to use this truststore instead of the default system one by specifying values for the properties for `trustStore` and `trustStorePassword`. These properties can be applied by modifying your service settings for PuppetDB and appending the required settings to the JAVA_ARGS variable. In Redhat the path to this file is `/etc/sysconfig/puppetdb`, in Debian `/etc/default/puppetdb`. For example:

```
# Modify this if you'd like to change the memory allocation, enable JMX, etc
JAVA_ARGS="-Xmx192m -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/var/log/puppetdb/puppetdb-oom.hprof -
Djavax.net.ssl.trustStore=/etc/puppetdb/ssl/truststore.jks -
Djavax.net.ssl.trustStorePassword=<PASSWORD>"
```

Note: Replace `<PASSWORD>` with the password found in

`/etc/puppetdb/ssl/puppetdb_keystore_pw.txt`.

Once this is done, you need to modify the database JDBC connection URL in your PuppetDB configuration as follows:

```
[database]
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //<HOST>:<PORT>/<DATABASE>?ssl=true
username = <USERNAME>
password = <PASSWORD>
```

Restart PuppetDB and monitor your logs for errors. If all goes well your connection should now be SSL.

Placing your own self-signed CA in a Java Keystore

If you so desire you can follow the documentation provided in the PostgreSQL JDBC SSL Client Setup instructions. This talks about how to generate a brand new SSL certificate, key and CA. Make sure you place your signed certificate and private key in the locations specified by the `ssl_cert_file` and `ssl_key_file` locations, and that you change the `ssl` setting to `on` in your `postgresql.conf`.

Once this is done you must modify the JDBC url in the database configuration section for PuppetDB. For example:

```
[database]
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //<HOST>:<PORT>/<DATABASE>?ssl=true
username = <USERNAME>
password = <PASSWORD>
```

Restart PuppetDB and monitor your logs for errors. If all goes well your connection should now be SSL.

Setting up SSL with a publicly signed certificate on the DB server

First, obtain your signed certificate using the process required by your commercial Certificate Authority. If you don't want to pay for individual certificates for each server in your enterprise you can probably get away with using wildcards in the subject or CN for your certificate. While at the moment DNS resolution based on CN isn't tested using the default SSLSocketFactory, we can not know if this will change going forward, and therefore if wildcard support will be included.

Use the documentation Secure TCP/IP Connections with SSL as this explains how to configure SSL on the server in detail. Make sure you place your signed certificate and private key in the locations specified by the `ssl_cert_file` and `ssl_key_file` locations, and that you change the `ssl` setting to `on` in your `postgresql.conf`.

Because the JDBC PostgreSQL driver utilizes the Java's system keystore, and because the system keystore usually contains all public CA's there should be no trust issues with the client configuration, all you need to do is modify the JDBC url as provided in the database configuration section for PuppetDB.

For example:

```
[database]
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //<HOST>:<PORT>/<DATABASE>?ssl=true
username = <USERNAME>
password = <PASSWORD>
```

Once this has been done, restart PuppetDB and monitor your logs for errors. If all goes well your connection should now be SSL.

Disabling SSL Verification

So before you commence, this is not recommended if you are wishing to gain the higher level of security provided with an SSL connection. Disabling SSL verification effectively removes the ability for the SSL client to detect man-in-the-middle attacks.

If however you really want this, the methodology is to simply modify your JDBC URL in the database configuration section of PuppetDB as follows:

```
[database]
classname = org.postgresql.Driver
subprotocol = postgresql
subname = //:/?ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory
username =
password =
```

Make this configuration change then restart PuppetDB and monitor your logs for errors. If all goes well your connection should now be SSL, and validation should be disabled.

PuppetDB 1.3 » 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 `<node>` 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 1.3 » Maintaining and Tuning

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

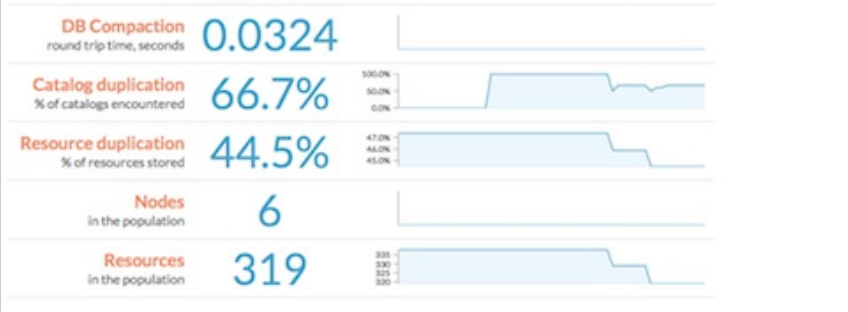
Monitor the Performance Dashboard

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. If you've used the PuppetDB module to install, you'll need to set the `listen_address` parameter. 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` 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).

Clean Up Old Reports

When the PuppetDB report processor is enabled on your Puppet master, PuppetDB will retain reports for each node for a fixed amount of time. This defaults to 14 days, but you can alter this to suit your needs using the `report-ttl` setting. The larger the value you provide for this setting, the more history you will retain; however, your database size will grow accordingly.

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 1.3 » Migrating Data

Migrating from ActiveRecord storeconfigs

If you're using exported resources with ActiveRecord storeconfigs, you may want to migrate your existing data to PuppetDB before connecting the master to it. This will ensure that whatever resources were being collected by the agents will still be collected, and no incorrect configuration

will be applied.

The existing ActiveRecord data can be exported using the `puppet storeconfigs export` command, which will produce a tarball that can be consumed by PuppetDB. Because this command is intended only to stop nodes from failing until they have check into PuppetDB, it will only include exported resources, excluding edges and facts.

NOTE: in order for this to work properly, you need to make sure you've run this command and generated the export tarball prior to configuring your master for PuppetDB.

Once you've run this command and generated an export tarball, you should follow the instructions below to import the tarball into your PuppetDB database.

Exporting data from an existing PuppetDB database

If you've been trying out PuppetDB using the embedded database and are ready to move to a production environment backed by PostgreSQL, or if you'd simply like to move your data from one PostgreSQL database to another one, you can use the `puppetdb-export` command (which is available in your `/usr/sbin` directory for versions of PuppetDB >= 1.2). All you'll need to do is run a command like this:

```
$ sudo puppetdb-export --outfile ./my-puppetdb-export.tar.gz
```

This command is intended to be run on the PuppetDB server, and assumes that PuppetDB is accepting plain-text HTTP connections on `localhost` port `8080` (which is PuppetDB's default configuration). If you've modified your PuppetDB HTTP configuration, you can specify a different hostname and port on the command line. For more info, run:

```
$ sudo puppetdb-export --help
```

While its not required it is recommended you run this tooling while there is no activity on your existing PuppetDB to ensure your data snapshot is consistent. Also the tool can put load on your production system, so you should plan for this before running it.

The generated tarball will contain a backup of all of your current catalog data (including exported resources) and all you report data. At this time fact data exporting is not supported.

Exporting data from a version of PuppetDB prior to 1.2

The `puppetdb-export` and `puppetdb-import` tools were added to PuppetDB in version 1.2. If you need to export data from an older version of PuppetDB, the easiest way to do so is to upgrade your existing PuppetDB to at least version 1.2 and then use the `puppetdb-export` tool.

Importing data to a new PuppetDB database

Once you have an export tarball and a new PuppetDB server up and running that you would like to import your data into, use the `puppetdb-import` command to do so. (This command is available in your `/usr/sbin` directory in versions of PuppetDB >= 1.2.) The syntax will look something like this:

```
$ sudo puppetdb-import --infile ./my-puppetdb-export.tar.gz
```

This command is intended to be run on the new PuppetDB server, and assumes that PuppetDB is accepting plain-text HTTP connections on `localhost` port `8080` (which is PuppetDB's default configuration). If you've modified your PuppetDB HTTP configuration, you can specify a different hostname and port on the command line. For more info, run:

```
$ sudo puppetdb-import --help
```

PuppetDB 1.3 » 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 1.3 » 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 = telnet
port = 8082
```

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 a telnet type REPL configured on port 8082:

```
$ telnet localhost 8082
Connected to localhost.
Escape character is '^]'.
;; Clojure 1.4.0
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 1.3 » Troubleshooting » KahaDB Corruption

What is KahaDB?

Internally PuppetDB utilises ActiveMQ for queuing commands received via the API and sometimes initiated internally. The queue today utilises a technology built for ActiveMQ called 'KahaDB' which is a file based persistence database designed specifically for high performance queuing.

The KahaDB storage for PuppetDB is located in a sub-directory underneath your configured `vardir` (see Configuration for more details). This sub-directory is generally, `mq/localhost/KahaDB`. For OSS PuppetDB the full path is usually `/var/lib/puppetdb/mq/localhost/KahaDB`.

Why does corruption occur?

In some cases this database may corrupt. Lots of things may cause this:

- Your disk may fill up, so writes are not finalised within the journal or database index.
- There might be a bug in the KahaDB code that the developers haven't catered for.

How do I recover?

During corruption, the simplest way to recover is to simply move the KahaDB directory out of the way and restart PuppetDB:

```
$ service puppetdb stop
$ cd /var/lib/puppetdb/mq/localhost
$ mv KahaDB KahaDB.old
$ service puppetdb start
```

(Note: it is very important for us that you preserve the old KahaDB directory. If the problem turns out to be something our Engineers haven't seen before we'll need that directory to replicate the problem, so make sure you preserve it.)

In most cases this is enough, however this means that any data that was not processed may be lost. This is usually only transient queue data however, and is not the persisted data that is stored in your PostgreSQL or HSQLDB database, so in most cases it is not a major concern. For most cases re-running puppet on your nodes will resubmit these lost commands for processing.

If these is going to be too destructive, then there is a few things you can do. But first of all, backup your KahaDB directory before doing anything so you can revert it after each attempt at the techniques listed below:

- You can try clearing your `db.data` file and recreating it. The `db.data` file represents your index, and clearing it may force it to be recreated from the logs.
- You can try clearing your `db-*.log` files. These files contain the journal and while KahaDB is usually good at finding pin-point corruption and ignoring these today (in fact much better since PuppetDB 1.1.0) there are still edge cases. Clearing them may let you skip over these bad blocks. It might be that only 1 of these files are corrupted, and the remainder are good so you could attempt clearing one at a time (newest first) to find the culprit.

How do I bring my corruption to the attention of developers?

In almost all cases though we want to hear about your corruption so we can improve the ways we deal with these problems. We would appreciate if you have these issues to look at our Bug Tracker for the term `kahadb` to see if you're problem is already known, and adding a comment if you see it yourself, including the version of PuppetDB you are using.

If the problem is unknown or new, make sure you log a new ticket including your `puppetdb.log` file, or at least the pertinent exception including the version of PuppetDB you are using and the potential cause of the corruption if you are aware of it. In all cases, make sure you preserve any backups of the `KahaDB` directory in its original corrupted state, this may be helpful to our Software Engineers to replicate the problem later.

PuppetDB 1.3 » API » Overview

Since PuppetDB collects lots of data from Puppet, it's an ideal platform for new tools and applications that use that data. You can use the HTTP API described in these pages to interact with PuppetDB's data.

Summary

PuppetDB's API uses a Command/Query Responsibility Separation (CQRS) pattern. This means:

- Data can be queried using a standard REST-style API. Queries are processed immediately.
- When making changes to data (facts, catalogs, etc), you must send an explicit command (as opposed to submitting data without comment and letting the receiver determine intent). Commands are processed asynchronously in FIFO order.

The PuppetDB API consists of the following parts:

- The REST interface for queries
- The HTTP command submission interface
- The wire formats that PuppetDB requires for incoming data

Queries

PuppetDB 1.3 supports versions 1 and 2 of the query API. Version 1 is backwards-compatible with PuppetDB 1.0.x, but version 2 has significant new capabilities, including subqueries.

PuppetDB's data can be queried with a REST API.

- Specification of the General Query Structure
- Available Operators
- Query Tutorial
- Curl Tips

The available query endpoints are documented in the pages linked below.

Query Endpoints

VERSION 2

Version 2 of the query API adds new endpoints, and introduces subqueries and regular expression operators for more efficient requests and better insight into your data. The following endpoints will continue to work for the foreseeable future.

- Facts Endpoint
- Resources Endpoint
- Nodes Endpoint
- Fact-Names Endpoint
- Metrics Endpoint

VERSION 1

Version 1 of the query API works with PuppetDB 1.1 and 1.0. It isn't deprecated, but we encourage you to use version 2 if you can.

In PuppetDB 1.0, you could access the version 1 endpoints without the `/v1/` prefix. This still works but is now deprecated, and we currently plan to remove support in PuppetDB 2.0. Please change your version 1 applications to use the `/v1/` prefix.

- Facts Endpoint
- Resources Endpoint
- Nodes Endpoint

- Status Endpoint
- Metrics Endpoint

EXPERIMENTAL

These endpoints are not yet set in stone, and their behavior may change at any time without regard for normal versioning rules. We invite you to play with them, but you should be ready to adjust your application on your next upgrade.

- Report Endpoint
- Event Endpoint

Commands

Commands are sent via HTTP but do not use a REST-style interface.

PuppetDB supports a relatively small number of commands. The command submission interface and the available commands are all described at the commands page:

- Commands (all commands, all API versions)

Unlike the query API, these commands are generally only useful to Puppet itself, and all format conversion and command submission is handled by the PuppetDB terminus plugins on your puppet master.

The "replace" commands all require data in one of the wire formats described below.

Wire Formats

All of PuppetDB's "replace" commands contain payload data, which must be in one of the following formats. These formats are also linked from the commands that use them.

- Facts wire format
- Catalog wire format
- Report wire format (experimental)

PuppetDB 1.3 » API » Query Tutorial

This page is a walkthrough for constructing several types of PuppetDB queries. It uses the version 2 API in all of its examples; however, most of the general principles are also applicable to the version 1 API.

If you need to use the v1 API, note that it lacks many of v2's capabilities, and be sure to consult the v1 endpoint references before attempting to use these examples with it.

How to Query

Queries are performed by performing an HTTP GET request to an endpoint URL and supplying a URL parameter called `query`, which contains the query to execute. Results are always returned in `application/json` form.

Queries are usually issued from code, but you can easily issue them from the command line using curl.

Querying with Curl

See “Curl Tips” for more detailed information about constructing curl commands.

Without SSL:

```
curl -H 'Accept: application/json' -X GET
http://puppetdb.example.com:8080/v2/resources --data-urlencode query@<filename>
```

This requires that PuppetDB be configured to accept non-SSL connections. By default, it will only accept unencrypted traffic from `localhost`.

With SSL:

```
curl -H 'Accept: application/json' -X GET
https://puppetdb.example.com:8081/v2/resources --cacert /etc/puppet/ssl/certs/ca.pem
--cert /etc/puppet/ssl/certs/thisnode.pem --key
/etc/puppet/ssl/private_keys/thisnode.pem --data-urlencode query@<filename>
```

This requires that you specify a certificate (issued by the same CA PuppetDB trusts), a private key, and a CA certificate.

In both examples, `<filename>` should be a file that contains the query to execute.

Resources Walkthrough

Our First Query

Let’s start by taking a look at a simple resource query. Suppose we want to find the user “nick” on every node. We can use this query:

```
["and",
  ["=", "type", "User"],
  ["=", "title", "nick"]]
```

This query has two `"="` clauses, which both must be true.

In general, the `"="` operator follows a specific structure:

```
["=", <attribute to compare>, <value>]
```

In this case, the attributes are “type” and “title”, and the values are “User” and “nick”.

The `"and"` operator also has a well-defined structure:

```
["and", <query clause>, <query clause>, <query clause>, ...]
```

The query clauses can be any legal query (including another `"and"`). At least one clause has to be specified, and all the clauses have to be true for the `"and"` clause to be true. An `"or"` operator is also available, which looks just like the `"and"` operator, except that, as you’d expect, it’s true if any specified clause is true.

The query format is declarative; it describes conditions the results must satisfy, not how to find them. So the order of the clauses is irrelevant. Either the type clause or the title clause could come

first, without affecting the performance or the results of the query.

If we execute this query against the `/resources` route, we get results that look something like this:

```
[{
  "parameters" : {
    "comment" : "Nick Lewis",
    "uid" : "1115",
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "allstaff",
    "home" : "/home/nick",
    "groups" : "developers",
    "ensure" : "present"
  },
  "sourceline" : 111,
  "sourcefile" : "/etc/puppet/manifests/user.pp",
  "exported" : false,
  "tags" : [ "firewall", "default", "node", "nick", "role::base", "users",
"virtual", "user", "account", "base", "role::firewall::office", "role",
"role::firewall", "class", "account::user", "office", "virtual::users",
"allstaff" ],
  "title" : "nick",
  "type" : "User",
  "resource" : "0ae7e1230e4d540caa451d0ade2424f316bfbf39",
  "certname" : "foo.example.com"
}]
```

Our results are an array of “resources”, where each resource is an object with a particular set of keys.

parameters: this field is itself an object, containing all the parameters and values of the resource sourceline: the line the resource was declared on sourcefile: the file the resource was specified in exported: true if the resource was exported by this node, or false otherwise tags: all the tags on the resource title: the resource title type: the resource type resources: this is an internal identifier for the resource used by PuppetDB certname: the node that the resource came from

There will be an entry in the list for every resource. A resource is specific to a single node, so if the resource is on 100 nodes, there will be 100 copies of the resource (each with at least a different certname field).

Excluding results

We know this instance of the user “nick” is defined on line 111 of /etc/puppet/manifests/user.pp. What if we want to check whether or not we define the same resource somewhere else? After all, if we’re repeating ourselves, something may be wrong! Fortunately, there’s an operator to help us:

```
["and",
  ["=", "type", "User"],
  ["=", "title", "nick"],
  ["not",
    ["and",
      ["=", "sourceline", "/etc/puppet/manifests/user.pp"],
      ["=", "sourcefile", 111]]]]
```

The `"not"` operator wraps another clause, and returns results for which the clause is not true. In this case, we want resources which aren’t defined on line 111 of /etc/puppet/manifests/user.pp.

Resource Attributes

So far we've seen that we can query for resources based on their `certname`, `type`, `title`, `sourcefile`, and `sourceline`. There are a few more available:

```
["and",
  ["=", "tag", "foo"],
  ["=", "exported", true],
  ["=", ["parameter", "ensure"], "present"]]
```

This query returns resources whose set of tags contains the tag "foo", and which are exported, and whose "ensure" parameter is "present". Because the parameter name can take any value (including that of another attribute), it must be namespaced using `["parameter", <parameter name>]`.

The full set of queryable attributes can be found in the resource endpoint documentation for easy reference.

Regular Expressions

What if we want to restrict our results to a certain subset of nodes? Certainly, we could do something like:

```
["or",
  ["=", "certname", "www1.example.com"],
  ["=", "certname", "www2.example.com"],
  ["=", "certname", "www3.example.com"]]
```

And this works great if we know exactly the set of nodes we want. But what if we want all the 'www' servers, regardless of how many we have? In this case, we can use the regular expression match operator `~`:

```
["~", "certname", "www\\d+\\.example\\.com"]
```

Notice that, because our regular expression is specified inside a string, the backslash characters must be escaped. The rules for which constructs can be used in the regexp depend on which database is in use, so common features should be used for interoperability. The regexp operator can be used on every field of resources except for parameters, and `exported`.

Facts Walkthrough

In addition to resources, we can also query for facts. This looks similar, though the available fields and operators are a bit different. Some things are the same, though. For instance, support you want all the facts for a certain node:

```
["=", "certname", "foo.example.com"]
```

This gives results that look something like this:

```
[ {
  "certname" : "foo.example.com",
  "name" : "architecture",
```

```
  "value" : "amd64"
}, {
  "certname" : "foo.example.com",
  "name" : "fqdn",
  "value" : "foo.example.com"
}, {
  "certname" : "foo.example.com",
  "name" : "hostname",
  "value" : "foo"
}, {
  "certname" : "foo.example.com",
  "name" : "ipaddress",
  "value" : "192.168.100.102"
}, {
  "certname" : "foo.example.com",
  "name" : "kernel",
  "value" : "Linux"
}, {
  "certname" : "foo.example.com",
  "name" : "kernelversion",
  "value" : "2.6.32"
} ]
```

Fact Attributes

In the last query, we saw that a “fact” consists of a “certname”, a “name”, and a “value”. As you might expect, we can query using “name” or “value”.

```
["and",
  ["=", "name", "operatingsystem"],
  ["=", "value", "Debian"]]
```

This will find all the “operatingsystem = Debian” facts, and their corresponding nodes. As you see, “and” is supported for facts, as are “or” and “not”.

Fact Operators

As with resources, facts also support the `~` regular expression match operator, for all their fields. In addition to that, numeric comparisons are supported for fact values:

```
["and",
  ["=", "name", "uptime_seconds"],
  [">=", "value", 100000],
  ["<", "value", 1000000]]
```

This will find nodes for which the uptime_seconds fact is in the half-open range [100000, 1000000). Numeric comparisons will always be false for fact values which are not numeric. Importantly, version numbers such as 2.6.12 are not numeric, and the numeric comparison operators can’t be used with them at this time.

Nodes Walkthrough

We can also query for nodes. Once again, this is quite similar to resource and fact queries:

```
["=", "name", "foo.example.com"]
```

The result of this query is:

```
["foo.example.com"]
```

This will find the node foo.example.com. Note that the results of a node query contain only the node names, rather than an object with multiple fields as with resources and facts.

Querying on Facts

Nodes can also be queried based on their facts, using the same operators as for fact queries:

```
["and",
  ["=", ["fact", "operatingsystem"], "Debian"],
  ["<", ["fact", "uptime_seconds"], 10000]]
```

This will return Debian nodes with uptime_seconds < 10000.

Subquery Walkthrough

The queries we've looked at so far are quite powerful and useful, but what if your query needs to consider both resources and facts? For instance, suppose you need the IP address of your Apache servers, to configure a load balancer. You could find those servers using this resource query:

```
["and",
  ["=", "type", "Class"],
  ["=", "title", "Apache"]]
```

This will find all the Class[Apache] resources, which each knows the certname of the node it came from. Then you could put all those certnames into a fact query:

```
["and",
  ["=", "name", "ipaddress"],
  ["or",
    ["=", "certname", "a.example.com"],
    ["=", "certname", "b.example.com"],
    ["=", "certname", "c.example.com"],
    ["=", "certname", "d.example.com"],
    ["=", "certname", "e.example.com"]]]
```

But this query is lengthy, and it requires some logic to assemble and run the second query. No, there has to be a better way. What if we could find the Class[Apache] servers and use the results of that directly to find the certname? It turns out we can, with this fact query:

```
["and",
  ["=", "name", "ipaddress"],
  ["in", "certname",
    ["extract", "certname", ["select-resources",
                             ["and",
                               ["=", "type", "Class"],
                               ["=", "title", "Apache"]]]]]
```

This may appear a little daunting, so we'll look at it piecewise.

Let's start with "select-resources". This operator takes one argument, which is a resource query, and returns the results of that query, in exactly the form you would expect to see them if you did a plain resource query.

We then use an operator called "extract" to turn our list of resources into just a list of certnames. So we now conceptually have something like

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

The "in" operator matches facts whose "certname" is in the supplied list. (For now, that list has to be generated from a subquery, and can't be supplied directly in the query, so if you want a literal list, you'll unfortunately still have to use a combination of "or" and "="). At this point, our query seems a lot like the one above, except we didn't have to specify exactly which certnames to use, and instead we get them in the same query.

Similarly, there is a "select-facts" operator which will perform a fact subquery. Either kind of subquery is usable from every kind of query (facts, resources, and nodes), subqueries may be nested, and multiple subqueries may be used in a single query. Finding use cases for some of those combinations is left as an exercise to the reader.

PuppetDB 1.3 » API » Query » 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
- fact-names
- nodes
- resources
- 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/v2/facts/<node>'
curl -H "Accept: application/json"
'http://localhost:8080/v2/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/v2/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/v2/nodes' --data-
urlencode 'query=["=", ["node", "active"], true]'
```

PuppetDB 1.3 » API » 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 object 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 object as outlined above. (Not to be confused with the `payload` field inside the command object.)
- 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.

The terminus plugins for puppet masters use this command API to update facts, catalogs, and reports for nodes.

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, as a JSON string, including the fields of the catalog wire format. Extra fields are ignored.

“replace catalog”, version 2

The payload is expected to be a Puppet catalog, as either a JSON string or an object, conforming exactly to the catalog wire format. Extra or missing fields are an error.

“replace facts”, version 1

The payload is expected to be a set of facts, as a JSON string, conforming to the fact wire format

“deactivate node”, version 1

The payload is expected to be the name of a node, as a JSON string, which will be deactivated effective as of the time the command is processed.

Experimental commands

“store report”, version 1

The payload is expected to be a report, containing events that occurred on Puppet resources. It is structured as a JSON object, confirming to the report wire format.

PuppetDB 1.3 » API » v2 » Query Structure

Summary

PuppetDB’s query API can retrieve data objects from PuppetDB for use in other applications. For example, the terminus plugins for puppet masters use this API to collect exported resources, and to translate node facts into the inventory service.

The query API is implemented as HTTP URLs on the PuppetDB server. By default, it can only be accessed over the network via host-verified HTTPS; see the jetty settings if you need to access the API over unencrypted HTTP.

API URLs

The first component of an API URL is the API version, written as `v1`, `v2`, etc. This page describes version 2 of the API, so every URL will begin with `/v2`. After the version, URLs are organized into a number of endpoints.

Endpoints

Conceptually, an endpoint represents a reservoir of some type of PuppetDB object. Each version of the PuppetDB API defines a set number of endpoints.

See the API index for a list of the available endpoints. Each endpoint may have additional sub-endpoints under it; these are generally just shortcuts for the most common types of query, so that you can write terser and simpler query strings.

Query Structure

A query consists of:

- An HTTP GET request to an endpoint URL…
- …which may or may not contain a query string as a `query` URL parameter…
- …and which must contain an `Accept: application/json` header.

That is, nearly every query will look like a GET request to a URL that resembles the following:

```
https://puppetdb:8081/v2/<ENDPOINT>?query=<QUERY STRING>
```

Query strings are optional for some endpoints, required for others, and prohibited for others; see each endpoint's documentation.

Query Strings

A query string must be:

- A URL-encoded…
- …JSON array, which may contain scalar data types (usually strings) and additional arrays…
- …which describes a complex comparison operation…
- …in prefix notation.

JSON arrays are delimited by square brackets (`[` and `]`), and items in the array are separated by commas. JSON strings are delimited by straight double-quotes (`"`) and must be UTF-8 text; literal double quotes and literal backslashes in the string must be escaped with a backslash (`"` is `\"` and `\` is `\\`).

"Prefix notation" means every array in a query string must begin with an operator, and the remaining elements in the array will be interpreted as that operator's arguments, in order. (The similarity to Lisp is intentional.)

A complete query string describes a comparison operation. When submitting a query, PuppetDB will check every possible result from the endpoint to see if it matches the comparison from the query string, and will only return those objects that match.

For a more complete description of how to construct query strings, see the Operators page.

Query Responses

All queries return data with a content type of `application/json`.

Tutorial and Tips

For a walkthrough on constructing queries, see the Query Tutorial page. For quick tips on using curl to make ad-hoc queries, see the Curl Tips page.

PuppetDB 1.3 » API » v2 » Query Operators

PuppetDB's query strings can use several common operators.

> Note: The operators below apply to version 2 of the query API. Not all of them are available to version 1 queries.

Binary Operators

Each of these operators accepts two arguments: a field, and a value. These operators are non-transitive: their syntax must always be:

```
["<OPERATOR>", "<FIELD>", "<VALUE>"]
```

The available fields for each endpoint are listed in that endpoint's documentation.

`=` (equality)

Matches if: the field's actual value is exactly the same as the provided value. Note that this does not coerce values — the provided value must be the same data type as the field. In particular, be aware that:

- Most fields are strings.
- Some fields are booleans.
- Numbers in resource parameters from Puppet are usually stored as strings, and equivalent numbers will not match — if the value of `someparam` were "0", then `["=", "someparam", "0.0"]` wouldn't match.

`>` (greater than)

Matches if: the field is greater than the provided value. Coerces both the field and value to floats or integers; if they can't be coerced, the operator will not match.

`<` (less than)

Matches if: the field is less than the provided value. Coerces both the field and value to floats or integers; if they can't be coerced, the operator will not match.

`>=` (less than or equal to)

Matches if: the field is greater than or equal to the provided value. Coerces both the field and value to floats or integers; if they can't be coerced, the operator will not match.

`<=` (greater than or equal to)

Matches if: the field is less than or equal to the provided value. Coerces both the field and value to floats or integers; if they can't be coerced, the operator will not match.

`~` (regexp match)

Matches if: the field's actual value matches the provided regular expression. The provided value must be a regular expression represented as a JSON string:

- The regexp must not be surrounded by the slash characters (`/rexegp/`) that delimit regexps in many languages.
- Every backslash character must be escaped with an additional backslash. Thus, a sequence like

`\d` would be represented as `\\d`, and a literal backslash (represented in a regexp as a double-backslash `\\`) would be represented as a quadruple-backslash (`\\\\`).

The following example would match if the `certname` field's actual value resembled something like `www03.example.com`:

```
["~", "certname", "www\\d+\\.example\\.com"]
```

Note: Regular expression matching is performed by the database backend, and the available regexp features are backend-dependent. For best results, use the simplest and most common features that can accomplish your task. See the links below for details:

- PostgreSQL regexp features
- HSQLDB (embedded database) regexp features

Boolean Operators

Every argument of these operators should be a complete query string in its own right. These operators are transitive: the order of their arguments does not matter.

`and`

Matches if: all of its arguments would match. Accepts any number of query strings as its arguments.

`or`

Matches if: at least one of its arguments would match. Accepts any number of query strings as its arguments.

`not`

Matches if: its argument would not match. Accepts a single query string as its argument.

Subquery Operators

Subqueries allow you to correlate data from multiple sources or multiple rows. (For instance, a query such as "fetch the IP addresses of all nodes with `Class[Apache]`" would have to use both facts and resources to return a list of facts.)

Subqueries are unlike the other operators listed above:

- The `in` operator results in a complete query string. The `extract` operator and the subqueries do not.
- An `in` statement must contain a field and an `extract` statement.
- An `extract` statement must contain a field and a subquery.

These statements work together as follows (working "outward" and starting with the subquery):

- The subquery collects a group of PuppetDB objects (specifically, a group of resources or a group of facts). Each of these objects has many fields.

- The `extract` statement collects the value of a single field across every object returned by the subquery.
- The `in` statement matches if the value of its field is present in the list returned by the `extract` statement.

Subquery	Extract	In
Every resource whose type is “Class” and title is “Apache.” (Note that all resource objects have a `certname` field, among other fields.)	Every `certname` field from the results of the subquery.	Match if the `certname` field is present in the list from the `extract` statement.

The complete `in` statement described in the table above would match any object that shares a `certname` with a node that has `Class[Apache]`. This could be combined with a boolean operator to get a specific fact from every node that matches the `in` statement.

`in`

An `in` statement constitutes a full query string, which can be used alone or as an argument for a boolean operator.

“In” statements are non-transitive and take two arguments:

- The first argument must be a valid field for the endpoint being queried.
- The second argument must be an `extract` statement, which acts as a list of possible values for the field.

Matches if: the field’s actual value is included in the list of values created by the `extract` statement.

`extract`

An `extract` statement does not constitute a full query string. It may only be used as the second argument of an `in` statement.

“Extract” statements are non-transitive and take two arguments:

- The first argument must be a valid field for the endpoint being subqueried (see second argument).
- The second argument must be a subquery.

As the second argument of an `in` statement, an `extract` statement acts as a list of possible values. This list is compiled by extracting the value of the requested field from every result of the subquery.

Available Subqueries

A subquery may only be used as the second argument of an `extract` statement, where it acts as a collection of PuppetDB objects. Each of the objects returned by the subquery has many fields; the `extract` statement takes the value of one field from each of those objects, and passes that list of values to the `in` statement that contains it.

In version 2 of the query API, the available subqueries are:

- `select-resources`
- `select-facts`

`SELECT-RESOURCES`

A `select-resources` subquery may only be used as the second argument of an `extract` statement.

It takes a single argument, which must be a complete query string which would be valid for the `/v2/resources` endpoint. (Note that `/v2/resources/<TYPE>` and `/v2/resources/<TYPE>/<TITLE>` cannot be directly subqueried.) Since the argument is a normal query string, it can itself include any number of `in` statements and subqueries.

`SELECT-FACTS`

A `select-facts` subquery may only be used as the second argument of an `extract` statement.

It takes a single argument, which must be a complete query string which would be valid for the `/v2/facts` endpoint. (Note that `/v2/facts/<NAME>` and `/v2/facts/<NAME>/<VALUE>` cannot be directly subqueried.) Since the argument is a normal query string, it can itself include any number of `in` statements and subqueries.

Subquery Examples

This query string queries the `/facts` endpoint for the IP address of all nodes with `Class[Apache]`:

```
["and",
  ["=", "name", "ipaddress"],
  ["in", "certname",
    ["extract", "certname",
      ["select-resources",
        ["and",
          ["=", "type", "Class"],
          ["=", "title", "Apache"]]]]]]
```

This query string queries the `/nodes` endpoint for all nodes with `Class[Apache]`:

```
["in", "name",
  ["extract", "certname",
    ["select-resources",
      ["and",
        ["=", "type", "Class"],
        ["=", "title", "Apache"]]]]]
```

This query string queries the `/facts` endpoint for the IP address of all Debian nodes.

```
["and",
  ["=", "name", "ipaddress"],
  ["in", "certname",
    ["extract", "certname",
      ["select-facts",
        ["and",
          ["=", "name", "operatingsystem"],
          ["=", "value", "Debian"]]]]]]
```

PuppetDB 1.3 » API » v2 » Querying Facts

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

Routes

`GET /v2/facts`

This will return all facts matching the given query. Facts for deactivated nodes are not included in the response. There must be an `Accept` header containing `application/json`.

URL PARAMETERS

- `query`: Required. A JSON array containing the query in prefix notation.

AVAILABLE FIELDS

- `"name"`: matches facts of the given name
- `"value"`: matches facts with the given value
- `"certname"`: matches facts for the given node

OPERATORS

See the Operators page

EXAMPLES

Using `curl` from localhost:

Get the operatingsystem fact for all nodes:

```
curl -X GET -H 'Accept: application/json' http://puppetdb:8080/v2/facts --data-
urlencode 'query=["=", "name", "operatingsystem"]'

[{"certname": "a.example.com", "name": "operatingsystem", "value": "Debian"},
 {"certname": "b.example.com", "name": "operatingsystem", "value": "RedHat"},
 {"certname": "c.example.com", "name": "operatingsystem", "value": "Darwin"},
```

Get all facts for a single node:

```
curl -X GET -H 'Accept: application/json' http://puppetdb:8080/v2/facts --data-
urlencode 'query=["=", "certname", "a.example.com"]'

[{"certname": "a.example.com", "name": "operatingsystem", "value": "Debian"},
 {"certname": "a.example.com", "name": "ipaddress", "value": "192.168.1.105"},
 {"certname": "a.example.com", "name": "uptime_days", "value": "26 days"}]
```

`GET /v2/facts/<NAME>`

This will return all facts for all nodes with the indicated name. There must be an `Accept` header containing `application/json`.

URL PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/facts` route, mentioned above. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

EXAMPLES

```
curl -X GET -H 'Accept: application/json'
http://puppetdb:8080/v2/facts/operatingsystem
```

```
[{"certname": "a.example.com", "name": "operatingsystem", "value": "Debian"},
 {"certname": "b.example.com", "name": "operatingsystem", "value": "Redhat"},
 {"certname": "c.example.com", "name": "operatingsystem", "value": "Ubuntu"}]
```

`GET /v2/facts/<NAME>/<VALUE>`

This will return all facts for all nodes with the indicated name and value. There must be an `Accept` header containing `application/json`.

URL PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/facts` route, mentioned above. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

EXAMPLES

```
curl -X GET -H 'Accept: application/json'
http://puppetdb:8080/v2/facts/operatingsystem/Debian

[{"certname": "a.example.com", "name": "operatingsystem", "value": "Debian"},
 {"certname": "b.example.com", "name": "operatingsystem", "value": "Debian}]
```

Response Format

Successful responses will be in `application/json`. Errors will be returned as non-JSON strings.

The result will be a JSON array, with one entry per fact. Each entry is of the form:

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

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

PuppetDB 1.3 » API » v2 » Querying Resources

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

Routes

`GET /v2/resources`

This will return all resources matching the given query. Resources for deactivated nodes are not included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Required. 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: [ {bool} {query}+ ] | [ "not" {query} ] | [ {match} {field} {value} ]
field:  string | [ string+ ]
value:  string
bool:   "or" | "and"
match:  "=" | "~"
```

`field` may be any of:

`tag`

> a case-insensitive tag on the resource

`certname`

> the name of the node associated with the resource

`[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 host except for “example.local” the JSON query structure would be:

```
["and", ["not", ["=", "certname", "example.local"]],
        ["=", "type", "File"],
        ["=", "tag", "magical"],
        ["=", ["parameter", "ensure"], "enabled"]
```

See the Operators page for the full list of available operators. Note that resource queries do not support inequality, and regexp matching is not supported against node status or parameter values.

`GET /v2/resources/<TYPE>`

This will return all resources for all nodes with the given type. Resources from deactivated nodes

aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/resources` route, mentioned above. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

EXAMPLES

Using `curl` from localhost:

```
curl -X GET -H "Accept: application/json"
'http://puppetdb:8080/v2/resources/User'

[{"parameters" : {
    "uid" : "1000,
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "1000,
    "home" : "/home/foo,
    "groups" : "users,
    "ensure" : "present"
  },
  "sourceline" : 10,
  "sourcefile" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "foo",
  "type" : "User",
  "certname" : "host1.mydomain.com"
}, {"parameters" : {
    "uid" : "1001,
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "1001,
    "home" : "/home/bar,
    "groups" : "users,
    "ensure" : "present"
  },
  "sourceline" : 20,
  "sourcefile" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "bar",
  "type" : "User",
  "certname" : "host2.mydomain.com"}]
```

`GET /v2/resources/<TYPE>/<TITLE>`

This will return all resources for all nodes with the given type and title. Resources from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/resources` route, mentioned above. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

EXAMPLES

Using `curl` from localhost:

```
curl -X GET -H "Accept: application/json"
'http://puppetdb:8080/v2/resources/User/foo'

[{"parameters" : {
    "uid" : "1000,
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "1000,
    "home" : "/home/foo,
    "groups" : "users,
    "ensure" : "present"
  },
  "sourceline" : 10,
  "sourcefile" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "foo",
  "type" : "User",
  "certname" : "host1.mydomain.com"
}]
```

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 1.3 » API » v2 » Querying Nodes

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

Routes

`GET /v2/nodes`

This will return all nodes matching the given query. Deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Required. A JSON array of query predicates, in prefix form, conforming to the format described below.

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

Only queries against `"name"` and facts are currently supported.

Fact terms must be of the form `["fact", <fact name>]`.

Node query supports all available operators. Inequality operators are only supported for fact queries, but regular expressions are supported for both name and facts.

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 nodes whose kernel is Linux and whose uptime is less than 30 days:

```
["and",
  ["=", ["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 hashes of the form:

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

The array is sorted alphabetically by `name`.

EXAMPLE

You can use `curl` to query information about nodes like so:

```
curl -H "Accept: application/json" 'http://localhost:8080/v2/nodes'
curl -G -H "Accept: application/json" 'http://localhost:8080/v2/nodes' --data-
urlencode 'query=["=", ["fact", "kernel"], "Linux"]'
```

`GET /v2/nodes/<NODE>`

This will return status information for the given node, active or not. There must be an `Accept` header containing `application/json`.

RESPONSE FORMAT

The response is the same format as for the /v1/status endpoint.

`GET /v2/nodes/<NODE>/facts`

This will return the facts for the given node. Facts from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics

are identical to the `query` parameter for the `/v2/facts` route. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

RESPONSE FORMAT

The response is the same format as for the /v2/facts endpoint.

`GET /v2/nodes/<NODE>/facts/<NAME>`

This will return facts with the given name for the given node. Facts from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/v2/facts` route. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

RESPONSE FORMAT

The response is the same format as for the /v2/facts endpoint.

`GET /v2/nodes/<NODE>/facts/<NAME>/<VALUE>`

This will return facts with the given name and value for the given node. Facts from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/v2/facts` route. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

RESPONSE FORMAT

The response is the same format as for the /v2/facts endpoint.

`GET /v2/nodes/<NODE>/resources`

This will return the resources for the given node. Resources from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Optional. A JSON array containing the query in prefix notation. The syntax and semantics are identical to the `query` parameter for the `/v2/resources` route. When supplied, the query is assumed to supply additional criteria that can be used to return a subset of the information normally returned by this route.

RESPONSE FORMAT

The response is the same format as for the /v2/resources endpoint.

`GET /v2/nodes/<NODE>/resources/<TYPE>`

This will return the resources of the indicated type for the given node. Resources from deactivated nodes aren't included in the response. There must be an `Accept` header containing

`application/json`.

This endpoint behaves identically to the `/v2/resources/<TYPE>` endpoint, except the resources returned include only those belonging to the node given in the URL for this route.

`GET /v2/nodes/<NODE>/resources/<TYPE>/<TITLE>`

This will return the resource of the indicated type and title for the given node. Resources from deactivated nodes aren't included in the response. There must be an `Accept` header containing `application/json`.

This endpoint behaves identically to the `/v2/resources/<TYPE>` endpoint, except the resources returned include only those belonging to the node given in the URL for this route.

PuppetDB 1.3 » API » v2 » Querying Fact Names

The `/fact-names` endpoint can be used to retrieve all known fact names.

Routes

`GET /fact-names`

This will return an alphabetical list of all known fact names, including those which are known only for deactivated nodes.

EXAMPLES

Using `curl` from localhost:

```
curl -X GET -H 'Accept: application/json' http://localhost:8080/v2/fact-names

["kernel", "operatingsystem", "osfamily", "uptime"]
```

Request

All requests must accept `application/json`.

Response Format

The response will be in `application/json`, and will contain an alphabetical JSON array containing fact names. Each fact name will appear only once, regardless of how many nodes have that fact.

```
[<fact>, <fact>, ..., <fact>, <fact>]
```

PuppetDB 1.3 » API » v2 » Querying Metrics

> Note: The v2 metrics endpoint is currently exactly the same as the v1 endpoint.

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 1.3 » API » v1 » 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

Successful responses will be in `application/json`. Errors will be returned as non-JSON strings.

The result is a JSON object containing two keys, “name” and “facts”. The “facts” entry is itself an

object mapping fact names to values:

```
{"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.

For example:

```
curl -X GET -H 'Accept: application/json'
http://puppetdb:8080/facts/a.example.com

{"name": "a.example.com",
  "facts": {
     "operatingsystem": "Debian",
     "ipaddress": "192.168.1.105",
     "uptime_days": "26 days"
  }
}
```

PuppetDB 1.3 » API » v1 » 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 1.3 » API » v1 » 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 1.3 » API » v1 » Querying Status

Routes

`GET /v1/status/nodes/<NODE>`

This will return status information for the given node. There must be an `Accept` header containing `application/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 1.3 » API » v1 » 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 1.3 » API » Experimental » Querying Reports

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

Routes

`GET /experimental/reports`

PARAMETERS

- `query`: Required. A JSON array of query predicates, in prefix form. (The standard `["<OPERATOR>", "<FIELD>", "<VALUE>"]` format.)

For example, for all reports run on the node with certname 'example.local', the JSON query structure would be:

```
["=", "certname", "example.local"]
```

OPERATORS

The only available OPERATOR is `=`.

FIELDS

The only available FIELD is `certname`, which is the name of the node associated with the report.

RESPONSE FORMAT

The response is a JSON array of report summaries for all reports that matched the input parameters. The summaries are sorted by the completion time of the report, in descending order:

```
[
  {
    "end-time": "2012-10-29T18:38:01.000Z",
    "puppet-version": "3.0.1",
    "receive-time": "2012-10-29T18:38:04.238Z",
    "configuration-version": "1351535883",
    "start-time": "2012-10-29T18:38:00.000Z",
    "id": "d4bcb35a-fb7b-45da-84e0-fceb7a1df713",
    "certname": "foo.local",
    "report-format": 3
    },
  {
    "end-time": "2012-10-26T22:39:32.000Z",
    "puppet-version": "3.0.1",
    "receive-time": "2012-10-26T22:39:35.305Z",
    "configuration-version": "1351291174",
    "start-time": "2012-10-26T22:39:31.000Z",
    "id": "5ec13ff5-c6fd-43fb-b5b1-59a00ec8e1f1",
    "certname": "foo.local",
    "report-format": 3
    }
]
```

EXAMPLE

You can use `curl` to query information about reports like so:

```
curl -G -H "Accept: application/json"
```

```
'http://localhost:8080/experimental/reports' --data-urlencode 'query=["=",
"certname", "example.local"]'
```

PuppetDB 1.3 » API » Experimental » Querying Events

Routes

`GET /experimental/events`

This will return all resource events matching the given query. (Resource events are generated from Puppet reports.) There must be an `Accept` header containing `application/json`.

PARAMETERS

- `query`: Required. A JSON array of query predicates, in prefix form (the standard `["<OPERATOR>", "<FIELD>", "<VALUE>"]` format), conforming to the format described below.

The `query` parameter is described by the following grammar:

```
query: [ {bool} {query}+ ] | [ "not" {query} ] | [ {match} {field} {value} ] |
[ {inequality} {field} {value} ]
field:          FIELD (conforming to [Fields](#fields) specification below)
value:          string
bool:           "or" | "and"
match:          "=" | "~"
inequality:     ">" | ">=" | "<" | "<="
```

For example, for all events in the report with id '38ff2aef3ffb7800fe85b322280ade2b867c8d27', the JSON query structure would be:

```
["=", "report", "38ff2aef3ffb7800fe85b322280ade2b867c8d27"]
```

To retrieve all of the events within a given time period:

```
["and", ["<", "timestamp", "2011-01-01T12:01:00-03:00"],
        [">", "timestamp", "2011-01-01T12:00:00-03:00"]]
```

To retrieve all of the 'failed' events for nodes named 'foo.*' and resources of type 'Service':

```
["and", ["=", "status", "failed"],
        ["~", "certname", "^foo\\."],
        ["=", "resource-type", "Service"]]
```

For more information on the available values for `FIELD`, see the fields section below.

- `limit`: Optional. If specified, this should be an integer value that will be used to override the `event-query-limit` configuration setting.

OPERATORS

See the Operators page for the full list of available operators. Note that inequality operators (`<`, `>`, `<=`, `>=`) are only supported against the `timestamp` FIELD.

FIELDS

`FIELD` may be any of the following. Unless otherwise noted, all fields support both equality and regular expression match operators, but do not support inequality operators.

`certname`

> the name of the node that the event occurred on.

`report`

> the id of the report that the event occurred in; these ids can be acquired via event queries or via the `/reports` query endpoint.

`status`

> the status of the event; legal values are `success`, `failed`, `noop`, and `skipped`.

`timestamp`

> the timestamp (from the puppet agent) at which the event occurred. This field supports the inequality operators. All values should be specified as ISO-8601 compatible date/time strings.

`resource-type`

> the type of resource that the event occurred on; e.g., `File`, `Package`, etc.

`resource-title`

> the title of the resource that the event occurred on

`property`:

> the property/parameter of the resource that the event occurred on; e.g., for a `Package` resource, this field might have a value of `ensure`. NOTE: this field may contain `NULL` values; see notes below.

`new-value`

> the new value that Puppet was attempting to set for the specified resource property. NOTE: this field may contain `NULL` values; see notes below.

`old-value`

> the previous value of the resource property, which Puppet was attempting to change. NOTE: this field may contain `NULL` values; see notes below.

`message`

> a description (supplied by the resource provider) of what happened during the event. NOTE: this field may contain `NULL` values; see notes below.

NOTES ON FIELDS THAT ALLOW `NULL` VALUES

In the case of a `skipped` resource event, some of the fields of an event may not have values. We handle this case in a slightly special way when these fields are used in equality (`=`) or inequality (`!=`)

queries; specifically, an equality query will always return `false` for an event with no value for the field, and an inequality query will always return `true`.

RESPONSE FORMAT

The response is a JSON array of events that matched the input parameters. The events are sorted by their timestamps, in descending order:

```
[
  {
    "certname": "foo.localdomain",
    "old-value": "absent",
    "property": "ensure",
    "timestamp": "2012-10-30T19:01:05.000Z",
    "resource-type": "File",
    "resource-title": "/tmp/reportingfoo",
    "new-value": "file",
    "message": "defined content as '{md5}49f68a5c8493ec2c0bf489821c21fc3b'",
    "report": "38ff2aef3ffb7800fe85b322280ade2b867c8d27",
    "status": "success"
  },
  {
    "certname": "foo.localdomain",
    "old-value": "absent",
    "property": "message",
    "timestamp": "2012-10-30T19:01:05.000Z",
    "resource-type": "Notify",
    "resource-title": "notify, yo",
    "new-value": "notify, yo",
    "message": "defined 'message' as 'notify, yo'",
    "report": "38ff2aef3ffb7800fe85b322280ade2b867c8d27",
    "status": "success"
  }
]
```

EXAMPLE

You can use `curl` to query information about events like so:

```
curl -G -H "Accept: application/json"
'http://localhost:8080/experimental/events' --data-urlencode 'query=["=",
"report", "38ff2aef3ffb7800fe85b322280ade2b867c8d27"]' --data-urlencode
'limit=1000'
```

PuppetDB 1.3 » API » 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"`. Version 2 of the "replace catalog" command will strictly validate this object and throw an error in the event of missing or extra fields. Version 1 of the "replace catalog" command will silently tolerate some inexact catalog objects.

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

The value of the `"metadata"` key must be `{ "api_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 four keys: `"name"`, `"version"`, `"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.

`"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 1.3 » API » 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.

PuppetDB 1.3 » API » Experimental » Report Wire Format, Version 1

> Note: This format is experimental, and may change at any time without regard for the normal API versioning rules.

Report interchange format

A report is represented as JSON (this implies UTF-8 encoding). Unless otherwise noted, `null` is not allowed anywhere in the report.

```
{
    "certname": <string>,
    "puppet-version": <string>,
    "report-format": <int>,
    "configuration-version": <string>,
    "start-time": <datetime>,
    "end-time": <datetime>,
    "resource-events": [<resource-event>, <resource-event>, ...]
}
```

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

`"certname"` is the certname the report is associated with.

`"puppet-version"` is the version of puppet that was run to generate this report.

`"report-format"` is the version number of the report format that puppet used to generate the original report data. This is a constant defined by puppet.

`"configuration-version"` is an identifier string that puppet uses to match a specific catalog for a node to a specific puppet run. This value is generated by puppet.

`"start-time"` is the time at which the puppet run began; see more details about the `datetime` format below.

`"end-time"` is the time at which the puppet run completed; see more details about the `datetime` format below.

Encoding

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

Data type: `<string>`

A JSON string. By virtue of the report being UTF-8, these must also be UTF-8.

Data type: `<integer>`

A JSON int.

Data type: `<datetime>`

A JSON string representing a date and time (with time zone), formatted based on the recommendations in ISO8601; so, e.g., for a UTC time, the String would be formatted as `YYYY-MM-DDThh:mm:ss.sssZ`. For non-UTC, the `Z` may be replaced with `±hh:mm` to represent the specific timezone.

Data type: `<resource-event>`

A JSON Object of the following form:

```
{
 "resource-type": <string>,
 "resource-title": <string>,
 "property": <string>,
 "timestamp": <datetime>,
 "status": <string>,
```

```
  "old-value": <string>,
  "new-value": <string>,
  "message": <string>
}
```

All keys are required.

`"resource-type"` is the name of the puppet resource type that the event occurred on.

`"resource-title"` is the title of the puppet resource that the event occurred on.

`"property"` is the name of the property of the resource for which the event occurred. This may be `null` only if the `status` is `skipped`.

`"timestamp"` is the date/time at which the event occurred.

`"status"` is a string representing the outcome of the event; this should be one of `success`, `failure`, or `skipped`.

`"old-value"` is the value that puppet determined the property to have held prior to the event.

`"new-value"` is the value that puppet was instructed to set the property to.

`"message"` is a descriptive message providing extra information about the event. This should be `null` if `status` is `success`.

Gaps with this wire format

1. Binary data needs to be dealt with (hopefully only for the `old-value` and `new-value` fields.□