Troubleshooting Puppet

Troubleshoot your Puppet infrastructure to fix problems quickly

Thomas Uphill
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Puppet Infrastructure'
- A synopsis of the book’s content
- More information on Troubleshooting Puppet
**About the Author**

**Thomas Uphill** is an RHCA who has been using Puppet since version 0.24. He has been a system administrator for nearly 20 years, more than 10 of which have been with Red Hat Linux and its derivatives. He wrote *Mastering Puppet, Packt Publishing*, a book on deploying Puppet in an Enterprise setting. He enjoys teaching others how to use Puppet to automate as much as possible. He runs the Seattle Puppet Users Group (PUGS) and volunteers with LOPSA and SASAG. You can find him at [http://ramblings.narrabilis.com](http://ramblings.narrabilis.com).
Preface

Puppet is a configuration management system that was written for system administrators so that they could manage a large number of systems efficiently and help maintain order. The deployment of Puppet becomes more complex as you increase the number of nodes in your environment. This book will help you troubleshoot your Puppet infrastructure to leverage your system's performance effectively by using different methods and techniques.

What this book covers

Chapter 1, Puppet Infrastructure, introduces the various components of a Puppet installation. We cover the problems that arise in the infrastructure and communication between the components.

Chapter 2, Writing Puppet Manifests, talks about manifests—the files in which you write the Puppet code. We look at some Puppet code, the common syntax, and formatting problems. We examine tools to help correct our code and discover how to use Git to automatically check our manifests. We also look at a few popular editors and how to configure them to help us edit Puppet code.

Chapter 3, Modules and Templates, looks at how code is organized in modules. We'll cover the non-manifest code that is also present in modules such as custom facts. We'll also look at templates and how to debug issues with them.

Chapter 4, Hiera and External Node Classifiers, is about Hiera, a hierarchical key:value lookup tool that helps Puppet keep the code and data distinct. An external node classifier (ENC) is a way of determining the classes that are applied to a node. In this chapter, we look at the problems that occur when configuring and using these tools.
Chapter 5, *The Marionette Collective*, examines MCollective, a Puppet Labs orchestration tool. ActiveMQ is the message broker that is used by MCollective. We cover problems with the configuration of both ActiveMQ and MCollective.

Chapter 6, *PuppetDB and Puppet Server*, describes how both these tools were written to deal with problems concerning scale. PuppetDB deals with exported resources and reporting and utilizes PostgreSQL. Puppet Server uses the TrapperKeeper framework that was developed by Puppet Labs and relies on a JVM and JRuby. In this chapter, we explore the problems with these tools and how to debug them using Ruby tools.

Chapter 7, *Help Me!*, looks at where you can go when you need some help. We introduce many of the community resources that are available to you when you cannot solve your Puppet problems.
As a Puppet engineer, you will encounter many different problems while working with Puppet. Puppet infrastructure comprises all the components that you can use to deploy Puppet on your nodes, and may include applications such as Apache, Passenger, Puppet Server, PuppetDB, and ActiveMQ. Knowing how the infrastructure works, and how the various components fit together and function, will help you troubleshoot your issues. An important aspect of how Puppet infrastructure works is the way the various components communicate with each other. Most of the communication between the components of your Puppet infrastructure is done via an HTTP/SSL REST API. Later in this chapter, we will use this API to troubleshoot our installation.

The first step towards understanding how all the components of a Puppet installation communicate is to know the different components. The components of a typical installation are shown in the following diagram:
In subsequent chapters, we will examine the workings of these various components in detail.

With this picture of the components in mind, we will now go through the main actions that take place during a Puppet run.

**The lifecycle of a Puppet run**

Communication between the nodes and the master in a Puppet environment is verified with a system of X.509 SSL certificates. The master operates as a certificate authority (CA) for the system, although you may specify another server to act as the CA. When the agent first runs on a node, there are several steps taken to set up the trust relationship between the node and the master, which are outlined as follows:

1. The agent contacts the master and downloads the CA certificate.
2. The agent generates a certificate for itself based on the `certname` configuration option, which is usually equivalent to the hostname of the node.
3. The agent issues a certificate signing request (CSR) to the master, asking the master to sign its certificate.
4. The master may choose to sign the certificate (if automatic signing is configured) or the operator of the master may sign the certificate.
5. The agent will check back every 2 minutes by default (configurable with the `waitforcert` option) to check whether its certificate has been signed.
6. Once it has been signed, the agent will download its signed certificate.

At this point, the trust relationship between the agent and master has been established. The subsequent Puppet runs will not have to perform these steps. These runs will have a different series of steps, which are outlined in the following list:

1. The agent contacts and informs the master about its operating environment. Environments are used to separate nodes into logical groups.
2. The master looks through all the available modules in the given environment and begins sending all the `/lib` subdirectories of the modules to the agent via the `pluginsync` method.
3. Once all the plugins have been downloaded to the node, the node runs the `facter` utility to generate a list of facts about the node. The agent then ships the fact list back to the master.
4. If you have PuppetDB configured in your environment, the facts are shipped to PuppetDB, which will decide to either create new entries for the node (if this is the first time, facts have arrived for this node), or update the existing fact values.

5. At this point, the master will use the fact list to generate values that are needed to compile the **catalog** for the node. A catalog is the living document of how a node is configured. It consists of all the Puppet modules, classes, and resources, that will be applied to a node.

6. The master will compile the catalog. This is the process of verifying that it is possible to apply resources to a node in a consistent manner. Puppet will generate a graph, with all the resources as vertices. If the graph has any cycles, then the compilation will fail. A cycle is also known as a circular dependency. It means that there are resources within the catalog that require each other, or that are mutually dependent. A failed compilation results in the agent exiting with an error code.

7. If the catalog compiles successfully, the catalog is then shipped to the agent.

8. At this point, the Puppet agent's run begins. All the resources in the catalog are applied sequentially to the node. When troubleshooting, it is important to remember that although the master is capable of running several catalog compilations at once, the agent on a node must be single-threaded by definition.

9. The catalog will either apply without error to the node, which is called a **clean run**, or fail and output errors. If the agent has a clean run and does not change any files on the node, the exit code of the process will be 0. If the agent has a clean run and does make any changes, the exit code will be 2. If the agent fails to apply all the resources to the node, which is called a **failed run**, the agent will exit with the exit code as either 4 (with no file changes) or 6 (with file changes).

   To have puppet agent return these exit codes, the **--detailed-exitcodes** option must be given when running the puppet agent. Note the test option, **-t**, includes this option.

10. Facter will be run at the end of a successful or clean run. The fact values that have changed or the new facts will be sent to PuppetDB. Note that if PuppetDB cannot be updated, the agent will mark the run as unclean.

11. Depending on your configuration, the agent will then generate a report, which will be shipped to the **report_server**. If reports are configured, then they will be sent for both failed and clean runs.
12. Several files on the node will also be updated at this point. When troubleshooting, it is often useful to examine the contents of the following files:

- The `classes.txt` file contains the list of classes in the catalog
- The `resources.txt` file contains a list of resources in the catalog
- The `state.yaml` file contains timestamps for various files on the system as well as scheduling information
- The `last_run_report.yaml` file contains all the log messages that would be output to the console or during the Puppet run (this may be overridden by the `--logdest` command-line argument)
- The `last_run_summary.yaml` file contains a summary of the last run, with a count of the resources that were changed, the time taken to complete the run, and additional metrics

Beyond initial communication issues between the agents and the master, the bulk of Puppet troubleshooting revolves around failed catalog compilation and application.

With an idea of how the agent runs, we can now look closely at how one can configure Puppet as we examine the `puppet.conf` file in the next section.

**Puppet configuration**

Configuration of both the Puppet master and the agents (nodes) is done with the same configuration file, `puppet.conf`. This file is located in different directories, which depend on the version of Puppet that you are running—the open source version, or the commercial version, Puppet Enterprise. The different locations are summarized in the following table:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Open source version</th>
<th>Puppet Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/Mac OS X</td>
<td><code>/etc/puppet/puppet.conf</code></td>
<td><code>/etc/puppetlabs/puppet/puppet.conf</code></td>
</tr>
<tr>
<td>Windows*</td>
<td><code>%PROGRAMDATA%\PuppetLabs\puppet\etc\puppet\puppet.conf</code></td>
<td></td>
</tr>
</tbody>
</table>

*Windows 2003 has a different location.*
You may also override the name and location of this file with the `config_file_name` and `config` options respectively. The `puppet.conf` configuration file uses the INI-style syntax, which consists of multiple sections. The `[main]` section is used for settings that apply to both the master and the agent modes of Puppet. The `[master]` section is for the settings that only affect the master, while the `[agent]` section is used to specify settings that are specific to the agent.

Here is a sample `puppet.conf` file:

```plaintext
[main]
logdir = /var/log/puppet
rundir = /var/run/puppet
ssldir = $vardir/ssl

[agent]
classfile = $vardir/classes.txt
localconfig = $vardir/localconfig
```

There are many more configuration options available. Puppet provides a utility for viewing all the available configuration options. To view all the available configuration options, use `puppet config print`. To view the options for a specific section, add `--section [section]` to the command, as shown in the following example in the `agent` section:

```bash
t@mylaptop ~ $ puppet config print --section agent |sort |head -10
agent_catalog_run_lockfile = /home/thomas/.puppet/var/state/agent_catalog_run.lock
agent_disabled_lockfile = /home/thomas/.puppet/var/state/agent_disabled.lock
allow_duplicate_certs = false
allow_variables_with_dashes = false
always_cache_features = false
archive_file_server = puppet
archive_files = false
async_storeconfigs = false
autoflush = true
autosign = /home/thomas/.puppet/autosign.conf
```
The important configuration options on the agent (when trying to troubleshoot) are those that are associated with communication with the master. By default, a node will look for a master named puppet. This is actually specified by the server option in the agent section. You can verify this setting with the following command:

t@mylaptop ~ $ puppet config print server --section agent

Another important option is the port from which one should contact the master. By default, it is port 8140, but you can change this with the masterport option. It is also possible to specify another server for the certificate (SSL) signing. This is specified by using the ca_server option.

As mentioned previously, the node will use the certname option to specify its own name when communicating with the master. When troubleshooting, it can be useful to specify a different certname option for a node in order to force the generation of a new certificate. You may also find it useful to specify the certname option with an appended domain, which is generally known as the fully qualified domain name (FQDN) of the node.

In summary, when you are troubleshooting the communication between the nodes and the master, the following options are important in determining the servers that will be contacted and the names that will be used in the communication:

- server: This is the name of the master server
- ca_server: This is the name of the CA server
- certname: This is the name of the node that has to be used in the certificate
- masterport: This is port 8140 by default

If you are new to the Puppet environment that you are troubleshooting, it is also useful to know the values of the following options:

- config_file_name: This is puppet.conf; this is rarely overridden
- confdir: This is the directory containing the configuration files of Puppet
- config: This is a combination of confdir/config_file_name
- vardir: This is a directory that contains variable files, and it has a value of /var/lib/puppet by default
- ssldir: This is the directory that contains the SSL certificates, and it has a value of $vardir/ssl by default
puppet help

Most commands on Unix-like operating systems provide a manual or man page. The man page provides information on the available options and general guidance on using the command. Puppet initially chose not to follow this standard and instead used the help argument when calling the puppet command to specify documentation. Recent versions of Puppet have included manual pages for specific subcommands of the Puppet command-line tool. To access an individual manual page, type man puppet-[subcommand]. For example, to access the manual page on using puppet help, use man puppet-help. You can also access the same manual page using puppet man help.

The available help topics for a recent edition of Puppet are shown in the following screenshot:

```
@yolaptop - $ puppet help
Usage: puppet <subcommand> [options] <action> [options]

Available subcommands:

  agent          The puppet agent daemon
  apply          Apply Puppet manifests locally
  ca             Local Puppet certificate authority management.
  catalog        Compile, save, view, and convert catalogs.
  cert           Manage certificates and requests
  certificates   Provide access to the CA for certificate management.
  certificate_request Manage certificate requests.
  certificate_revocation_list Manage the list of revoked certificates.
  config         Interact with Puppet's settings.
  describe       Display help about resource types
  device         Manage remote network devices
  ddc            Generate Puppet documentation and references
  facts          Retrieve and store facts
  file           Retrieve and store files in a filebucket
  filebucket     Store and retrieve files in a filebucket
  help           Display Puppet help
  inspect        Send an inspection report
  instrumentation_data Manage instrumentation listener accumulated data. DEPRECATED.
  instrumentation_listener Manage instrumentation listeners. DEPRECATED.
  instrumentation_probe Manage instrumentation probes. DEPRECATED.
  key            Create, save, and remove certificate keys.
  kick           Remotely control puppet agent
  pan            Display Puppet visual pages.
  master         The puppet master daemon
  module         Create, installs and searches for modules on the Puppet Forge.
  node           View and manage node definitions.
  parser         Interact directly with the parser.
  plugin         Interact with the Puppet plugin system.
  proxy          deprecated querying daemon for asynchronous stores
  resource       The resource abstraction layer shell
  resource_type  View classes, defined resource types, and nodes from all manifests.
  secret_agent   Unlatch puppet agent
  status         View puppet server status.

See 'puppet help <subcommand> <action>' for help on a specific subcommand action.
See 'puppet help <subcommand>' for help on a specific subcommand.
```

Puppet v3.7.4
@yolaptop - $
For more information about a specific command, issue the command after puppet help. For example, for more information on puppet config, use puppet help config. This is demonstrated in the following code:

t@mylaptop ~ $ puppet help config

 USAGE: puppet config<action> [-section SECTION_NAME]

This subcommand can inspect and modify settings from Puppet's 'puppet.conf' configuration file. For documentation about individual settings, see http://docs.puppetlabs.com/references/latest/configuration.html.

OPTIONS:
--render-as FORMAT - The rendering format to use.
--verbose         - Whether to log verbosely.
--debug           - Whether to log debug information.
--section SECTION_NAME - The section of the configuration file to interact with.

ACTIONS:
print   Examine Puppet's current settings.
set     Set Puppet's settings.

See 'puppet man config' or 'man puppet-config' for full help.

You can also add subcommands to have even more specific information returned by the command, as follows:

t@mylaptop ~ $ puppet help config print

 USAGE: puppet config print [--section SECTION_NAME] (all | <setting> [<setting> ...])

Prints the value of a single setting or a list of settings.
OPTIONS:
--render-as FORMAT - The rendering format to use.
--verbose - Whether to log verbosely.
--debug - Whether to log debug information.
--section SECTION_NAME - The section of the configuration file to interact with.

See 'puppet man config' or 'man puppet-config' for full help.

The output of puppet help shows all the available arguments for the Puppet command-line utility. We will now see the usefulness of a selection of these arguments.

puppet resource
Using puppet resource can be a valuable troubleshooting tool. When you are diagnosing a problem, puppet resource can be used to inspect a node and verify how Puppet sees the state of a resource. For example, if we had a Linux node running an SSH daemon (sshd), we could ask Puppet about the state of the sshd service using puppet resource, as follows:

```
t@mylaptop ~ $ sudo puppet resource service sshd
service { 'sshd':
  ensure => 'running',
  enable => 'true',
}
```

As you can see, Puppet returned the current status of the service in Puppet code. By using puppet resource, we can query the node for any resource type that is known to Puppet. For instance, we can do this to view the status of the bind package of a node, as follows:

```
t@mylaptop ~ $ sudo puppet resource package bind
package { 'bind':
  ensure => 'absent',
}
```
We can also inspect the settings for a file, as follows:

t@mylaptop ~ $ sudo puppet resource file /etc/resolv.conf
file { '/etc/resolv.conf':
  ensure => 'file',
  content => '{md5}463bd26e077bc01a9368378737ef5bf0',
  ctime => '2015-03-02 21:04:21 -0800',
  group => '0',
  mode => '644',
  mtime => '2015-03-02 21:04:21 -0800',
  owner => '0',
  selrange => 's0',
  selrole => 'object_r',
  seltype => 'net_conf_t',
  seluser => 'system_u',
  type => 'file',
}

**puppet apply**

When troubleshooting, it can often be useful to apply a small chunk of code rather than a whole catalog. By using `puppet apply`, you can specify a manifest file, which can be applied directly to a node. For example, to create a file named `/tmp/trouble` on the local node with the content `Hello, Troubleshooter!`, create the following manifest file named `trouble.pp`:

```puppet
file {'/tmp/trouble':
  content => "Hello, Troubleshooter!\n"
}
```

When we run `puppet apply` on this manifest, Puppet will create the `/tmp/trouble` file as expected:

t@mylaptop ~ $ puppet apply trouble.pp

Notice: Compiled catalog for mylaptop in environment production in 0.21 seconds

Notice: /Stage[main]/Main/File[/tmp/trouble]/ensure: defined content as '{md5}7b6223913d48607e89a7c2f117444d0'

Notice: Finished catalog run in 0.03 seconds

t@mylaptop ~ $ cat /tmp/trouble
Hello, Troubleshooter!
When troubleshooting, it can be useful to add the `--debug` option when running `puppet apply`. Puppet will print information about how facts were compiled for the node, in addition to debugging information related to the application of resources.

**puppet parser validate**

The command-line utility can also be used to verify the syntax of your manifests. This can be useful when trying to find an issue with the compilation of your catalog. You can verify individual files by adding them as arguments to the command. For instance, the following manifest has a syntax error:

```ruby
file {'bad':
  ensure => 'directory',
  path   => '/tmp/bad'
  owner  => 'root',
}
```

We can verify this with `puppet parser validate`, which shows the following error:

```
t@mylaptop ~/trouble/01 $ puppet parser validate bad.pp
Error: Could not parse for environment production: Syntax error at 'owner'; expected '}' at /home/thomas/trouble/01/bad.pp:4
```

As you can see, there should be a comma after `/tmp/bad` since there is another attribute specified for the file resource.

I find myself using this command often enough to use the `ppv` alias for `puppet parser validate`.

**Log files and the catalog**

Logging in Puppet can be enabled on a client node (agent) with the `--debug` option to Puppet agent. This will output a lot of information. Each plugin file will be displayed as it is being read and executed. Once the catalog compiles, as each resource is applied to the machine, debugging information will be shown on the agent.

However, when you are debugging, your catalog may fail to compile. If this is the case, then you will need to examine the logs on the master. Where the logs are kept on the master depends on the way you have your master configured. The Puppet master process can be run either through a Ruby HTTP library named WEBrick, or via Passenger on a web server such as Apache or Nginx. Also, a third option now exists. You can also use the `puppetserver` application, which is a combination of JRuby and Clojure.
**puppet master**
Both the WEBrick and Passenger methods of running a Puppet master are equivalent to running `puppet master` from the command line. The configuration options for the Puppet master can be viewed with `puppet help master`.

By default, Puppet will log using `syslog` to the system logs (usually `/var/log/messages`). You can change this by making the `--logdest` option point at a file (logdest is used to specify the destination for log files, logdest may one of syslog, console, or the path to a file). If you are running the WEBrick server, then you can start the server like this:

```
# puppet master --logdest /var/log/puppet/master.log
```

When using Passenger, you will have a `config.ru` file, which is installed with the `puppet-passenger` package. You can add the additional logging options to this file.

To enable the debugging of logs, add the `--debug` option in addition to the `--logdest` option. You may also enable the `--verbose` option.

**puppetserver**
puppetserver is the new server for Puppet that is based on the server for PuppetDB. It uses a Java Virtual Machine (JVM) to run JRuby for the Puppet Server. This mechanism also uses Clojure. Puppet Labs has already made puppetserver the default Puppet master implementation for the new installations of Puppet Enterprise. The configuration of puppetserver is different from the Puppet master configuration. The server is configured by files that are located in `/etc/puppetserver` by default. Since the server is running through a JVM, it uses the Logback library. The configuration for Logback is in `/etc/puppetserver/logback.xml`. To enable debug logs, edit this file and change the log level from `info` to `debug`, as follows:

```xml
<root level="debug">
  <!--<appendener-ref ref="STDOUT"/>-->
  <appendener-ref ref="${logappender:-DUMMY}" />
  <appendener-ref ref="F1"/>
</root>
```

Changes made to this file are recognized immediately by the server. There is no need to restart the service. For entirely too much information, try setting the level to `trace`. More information on puppetserver can be found at `https://github.com/puppetlabs/puppet-server`. 
When the catalog is compiled for a node, the master will send the catalog to the node. The node will store the catalog in the client_data subdirectory of /var/lib/puppet. The catalog will be in the JSON format (previous versions of Puppet used the YAML format for catalogs). Reading the JSON files is not as simple as reading the YAML files. A tool that can help make things easier is jq, a command-line JSON processor. You can use it to search through the JSON files. For instance, to view the classes within a catalog, use the following:

```
$ jq '.data.classes[]' <hostname.json
```

To view the resources defined in the catalog, use the following command:

```
$ jq '.data.resources[]' <hostname.json
```

To filter out the resources tagged with a specific tag, use "class" as follows:

```
jq '.data.resources[] | select(.tags[] == "class")' <hostname.json
```

Once you learn the syntax for jq, searching through large catalogs becomes an easy task. This can help you find out where your resources are defined quickly. More information on jq can be found at http://stedolan.github.io/jq/.

**Communication issues**

Before you can begin debugging complex catalog problems, you need your nodes to communicate with each other. Communication problems between nodes and the master can either be network-related or certificate-related (SSL).

**Network-related problems**

When the Puppet agent is started on a node, one of the first things that the agent does is look up the value for the server option. You can either specify this with --server on the command line, or with server=[hostname] in the puppet.conf configuration file. By default, Puppet will look for a server named puppet. If it cannot find one named puppet, it will then try puppet.[your domain].

What Puppet believes to be your domain may be obtained by running facter domain.
When you are debugging the initial communication problems, you need to first verify that your nodes can find the Puppet master. For Unix systems, the way in which the system searches for a machine by name is called the `gethostbyname` system call. This system call uses the Name Service Switch (NSS) library to find a host in a number of databases. NSS is configured by the `/etc/nsswitch.conf` file. The line in this file that is used to find hosts by their respective names is the hosts line. The default configuration on most of the systems is the following:

```plaintext
hosts:  files dns
```

This line means that the system will search for hosts by name in the local files first. Then, if the host is not found, it will search in the Internet Domain Name System (DNS). The local file that is first consulted is `/etc/hosts`. This file contains static host entries. If you inherited your Puppet environment, you should look in this file for statically defined Puppet entries. If the machine `puppet` or `puppet.[domain]` is not found in `/etc/hosts`, the system then queries the DNS to find the host. The DNS is configured with the `/etc/resolv.conf` file on the Unix systems.

When troubleshooting, be aware that the `domain` fact is calculated using a combination of calls to the utility `hostname` and looking for a domain line in `/etc/resolv.conf`.

This file is known as the resolver configuration file. It's important to verify that you can reach the servers listed in the `nameserver` lines in this file. Your file may contain a search line. This line lists the domains that will be appended to your search queries. Consider a situation where the search line is as follows:

```plaintext
search example.com external.example.com internal.example.com
```

When you search for Puppet, the system will first search for `puppet`, then `puppet.example.com`, then `puppet.external.example.com`, and finally `puppet.internal.example.com`.

Several utilities exist for the testing of DNS. Among these utilities, `host` and `dig` are the most common. An older utility, `nslookup`, may also be used. To lookup the `ipaddress` option of the default Puppet Server, use the following:

```plaintext
t@mylaptop ~ $ host puppet
Host puppet not found: 3(NXDOMAIN)
```
In this example, the host puppet is not found. Yet, I know that this node works as expected. Remember that the system uses the `gethostbyname` system call when looking up the Puppet Server. Another utility on the system uses this call—the `ping` utility. When we try to ping the Puppet Server, this succeeds, and the output is as follows:

```
t@mylaptop ~ $ ping -c 1 puppet
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.093 ms
--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.093/0.093/0.093/0.000 ms
```

As you can see, the loopback address (127.0.0.1) is being used for the Puppet Server. We can verify that this information is coming from the `/etc/hosts` file using `grep`:

```
t@mylaptop ~ $ grep puppet /etc/hosts
127.0.0.1 localhost localhost.localdomain mylaptop localhost4
localhost4.localdomain4 mylaptop.example.com puppet.example.com puppet
```

Remembering the difference between using `host` or `dig` and using the `gethostbyname` system call can quickly help you find problems with your configuration. Adding an entry to `/etc/hosts` for your Puppet Server also bypasses any DNS problems that you may have in the initial configuration of your nodes.

**Netcat**

The next step in diagnosing network issues is verifying that you can reach the Puppet Server on the `masterport`, which is by default TCP port 8140. The `masterport` number may be changed, though. So, you should first confirm the port number using `puppet config print masterport`. One of the simplest tests to verify that you can reach the Puppet Server on port 8140 is to use **Netcat**. Netcat is known as the Swiss Army knife of network tools. You can do many interesting things with Netcat. More information about Netcat is available at [http://nmap.org/ncat/](http://nmap.org/ncat/).

There are several versions of Netcat available. The version installed on the most recent distributions is Ncat. The rewrite was done by Nmap (for more information, visit [https://nmap.org](https://nmap.org)).
To verify that you can reach port 8140 on your Puppet Server, issue the following command:

```bash
# nc -v puppet 8140
```

Connection to puppet 8140 port [tcp/*] succeeded!

If your Puppet Server was inaccessible, you will see an error message that looks like this:

```
nc: connect to puppet port 8140 (tcp) failed: Connection refused
```

If you see a `Connection refused` error as in the preceding output, this may indicate that there is a host-based firewall on the Puppet Server that is refusing the connection. Connection refusal means that you were able to contact the server, but the server did not permit the communication on the specified port. The first step in troubleshooting this type of problem is to verify that the Puppet Server is listening for connections on the port. The `lsof` utility can do this for you, as shown in the following code:

```
[root@puppet ~]# lsof -i :8140
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
java    1960 puppet   18r  IPv6  22323      0t0  TCP *:8140 (LISTEN)
```

My Puppet Server is running the `java` process because `puppetserver` runs inside a JVM. We see `java` as the process name in the `lsof` output. If you do not see any output here, then you will know that your Puppet Server is not listening on the 8140 port.

If you do see a line with the `LISTEN` text, then your Puppet Server is listening and a firewall is blocking the communication. Host-based firewalls on Linux are configured with the `firewalld` system or `iptables`, depending on your distribution. More information on these two systems can be found at [http://en.wikipedia.org/wiki/Iptables](http://en.wikipedia.org/wiki/Iptables) and [https://fedoraproject.org/wiki/FirewallD](https://fedoraproject.org/wiki/FirewallD).

Ubuntu distributions also include an Uncomplicated Firewall (ufw) utility to configure iptables. BSD-based systems will use the Berkeley Packet Filter (pf) or IPFilter. Knowing how to configure your host-based firewall configuration is a key troubleshooting skill.
If you are familiar with firewall configuration, you can add port 8140 to the allow list and solve the problem. If you are new to firewall configuration, you may choose to temporarily disable the firewall to aid your troubleshooting. Although a perimeter firewall is often a better solution, host-based firewalls should be used wherever possible to avoid accidentally or unintentionally exposing ports on your servers. When you have fixed the problem, turn the host-based firewall back on. On an Enterprise Linux-based distribution, the following will disable the host-based firewall:

```
[root@puppet state]# service iptables stop
iptables: Setting chains to policy ACCEPT: filter [  OK  ]
iptables: Flushing firewall rules: [  OK  ]
iptables: Unloading modules: [  OK  ]
```

If removing your host-based firewall does not solve your communication issue and you have verified that the service is listening on the correct port, then you will have to resort to advanced network troubleshooting tools.

Tools that may help in this case are `mtr` and `traceroute`. It is important to note that, even if a ping test fails, you may still be able to reach your Puppet Server on the masterport. The ping utility uses ICMP packets, which may be blocked or restricted on your network. If the `netcat` test still fails after addressing the firewall concerns, then you should try the `mtr` utility to check whether you can find where your communication is not reaching the server. For example, to test connectivity with the puppet server, issue the following command:

```
# mtr puppet
```

As an example, from my laptop, the following is the `mtr` output when attempting to reach https://puppetlabs.com:

```
```

<table>
<thead>
<tr>
<th>Host</th>
<th>Last</th>
<th>Avg</th>
<th>Best</th>
<th>Wrt</th>
<th>StDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00.wt</td>
<td>23.1</td>
<td>4.8</td>
<td>1.5</td>
<td>28.1</td>
<td>3.9</td>
</tr>
<tr>
<td>2.777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. te-0-1-0-22-sur03.bellevue.wa.seattle.comcast.net</td>
<td>21.8</td>
<td>15.4</td>
<td>11.2</td>
<td>42.7</td>
<td>5.6</td>
</tr>
<tr>
<td>4. be-40-ar01.wa.seattle.comcast.net</td>
<td>14.6</td>
<td>17.5</td>
<td>12.5</td>
<td>32.1</td>
<td>4.1</td>
</tr>
<tr>
<td>4. bc-1-3-0-0-10-cp01.seattle.wa.ibm.comcast.net</td>
<td>13.2</td>
<td>29.1</td>
<td>13.0</td>
<td>106.6</td>
<td>9.6</td>
</tr>
<tr>
<td>5. be-2-19-0-0-cp01.miami.fl.ibone.comcast.net</td>
<td>20.2</td>
<td>18.9</td>
<td>13.0</td>
<td>36.6</td>
<td>5.5</td>
</tr>
<tr>
<td>6. as3050-1-c.seattle.wa.ibmone.comcast.net</td>
<td>13.8</td>
<td>19.3</td>
<td>12.8</td>
<td>47.1</td>
<td>7.1</td>
</tr>
<tr>
<td>8. en0.bhr69-eq01.dem01.networklayer.com</td>
<td>45.6</td>
<td>44.6</td>
<td>39.2</td>
<td>91.4</td>
<td>7.7</td>
</tr>
<tr>
<td>9. pe15.dsr69-eq01.dem01.networklayer.com</td>
<td>65.1</td>
<td>67.3</td>
<td>60.9</td>
<td>150.3</td>
<td>9.5</td>
</tr>
<tr>
<td>10. pe25.dsr69.dllst3.networklayer.com</td>
<td>66.2</td>
<td>68.1</td>
<td>66.7</td>
<td>106.4</td>
<td>12.8</td>
</tr>
</tbody>
</table>

---

[17]
If you were unable to reach the Puppet Server, the last line in the host list would be `???.` The line immediately preceding the `???,` line would be the point at which the line of communication between the node and master was broken.

After you have verified that the network communication between the node and master is working as expected, the next issue that you should resolve is certificates.

## SSL-related problems

Puppet uses X509 certificates to secure the communication between nodes and the master. As a Puppet administrator, you should know how the SSL certificates and a CA works.

Your infrastructure may have a separate server that acts as a CA for your Puppet installation. The CA is the certificate that is used to sign all the certificates that are generated by your master(s). If your CA is a separate server, the `ca_server` option will be specified in the `puppet.conf` file.

Although the server may be specified from the command line when running `puppet agent`, the `ca_server` option cannot.

By default, the CA certificate is generated on the first run of either the Puppet master or `puppetserver`. The certificate is stored in `/var/lib/puppet/ssl/ca/ca_crt.pem` for the Open Source Puppet (OSS) or `/etc/puppetlabs/puppet/ssl/ca/ca_crt.pem` for Puppet Enterprise (PE). To view the information in the certificate, use OpenSSL's `x509` utility, as follows:

```bash
# openssl x509 -in ca_crt.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Puppet CA: puppet.example.com
        Validity
            Not Before: Feb 28 06:29:29 2015 GMT
            Not After : Feb 28 06:29:29 2020 GMT
        Subject: CN=Puppet CA: puppet.example.com
        Subject Public Key Info:
```
Public Key Algorithm: rsaEncryption

Public-Key: (4096 bit)

Modulus:

00:99:2f:50:c4:5a:9c:e9:3a:4a:f0:1b:9b:9e:d1:

... 

If you are new to the openssl command-line utility, try running openssl help (help is not actually an option, but it will cause the openssl command to print helpful information). Each of the subcommands to the openssl utility has its own Unix manual page. The manual page for the x509 subcommand can be found using man x509.

The preceding information shows that the CA certificate was automatically generated and has a five-year expiry. 5 years has been the default expiry time for some time now, and many Puppet installations are nearly 5 years old and require the generation of new CA certificates. If everything suddenly stopped working, you may wish to verify the expiry date of your CA. In addition to the expiry time, we can see the subject of the certificate, puppet.example.com. This is the name that Puppet has given to the CA based on the hostname and domain facts when the master/Puppet Server was started.

If you are diagnosing a certificate issue, you can first start by downloading the CA certificate. This can be done with the curl or wget utilities. In this example, we will use curl and pass the --insecure option to curl (since we have not downloaded the CA yet and cannot verify the certificate at this point), as follows:

```
$ curl --insecure https://puppet:8140/production/certificate/ca
```

```
-----BEGIN CERTIFICATE-----
MIIFfjCCA2agAwIBAgIBATANBgkqhkiG9w0BAQsFADAoMSYwJAYDVQQDDB1QdXBw
ZXQgQ0E6IHB1cHBldC5leGFtcGxlLmNvbTAeFw0xNTAyMjgwNjI5MjlaFw0yMDAy
...
```

We can use a pipe (|) to direct the curl output to openssl and verify the certificate, as follows:

```
$ curl --insecure https://puppet:8140/production/certificate/ca | openssl x509 -text
```

```
% Total  % Received % Xferd  Average Speed Time     Time     Time/Total Current
Dload  Upload   Total   Spent    Left  Speed
```
Puppet Infrastructure

100 1964 100 1964 0 0 6684 0 -:--:-- --:--:-- --:--:--
- 6680

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=Puppet CA: puppet.example.com
    Validity
      Not Before: Feb 28 06:29:29 2015 GMT
      Not After : Feb 28 06:29:29 2020 GMT
    Subject: CN=Puppet CA: puppet.example.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (4096 bit)
      Modulus:
        00:99:2f:50:c4:5a:9c:e9:3a:4a:f0:1b:9b:9e:d1:
    ...

If the CA certificate verifies correctly, the next step is to attempt to retrieve the certificate for your node. You can do this by first downloading the CA certificate to a local file as follows:

$ curl --insecure https://puppet:8140/production/certificate/ca > ca_crt.pem

In this example, my hostname is mylaptop. I will attempt to download my certificate from the master using curl (verifying the communication with the previously downloaded CA certificate):

$ curl --cacert ca_crt.pem https://puppet:8140/production/certificate/mylaptop

------BEGIN CERTIFICATE-----
MIIFcTCCAlmgAwIBAgIBBiBDBANBgkqhkiG9w0BAQsFAdAOuMSYwJAYDVQQD
ZXQgQ0B6IHBlcHBldC51eGFtcGxlMvbTaeFw0xNTAzMDExNjMzMDdaFw0yMDAy
------END CERTIFICATE-----
As you can see, this succeeded. If we pipe the output to OpenSSL, we see that the subject of the certificate is `mylaptop` and the certificate has not expired:

```
$ curl --cacert ca_crt.pem https://puppet:8140/production/certificate/mylaptop | openssl x509 -text
% Total    % Received % Xferd  Average Speed   Time    TimeTime
Current
Dload  Upload Total   Spent    Left  Speed
100  1948  100 1948 0     0  6155 0  --:--:--  --:--:--  --:--:--  --:--:--
-6145
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 4 (0x4)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=Puppet CA: puppet.example.com
    Validity
      Not Before: Mar 1 06:33:07 2015 GMT
      Not After : Feb 29 06:33:07 2020 GMT
    Subject: CN=mylaptop
...
```

Since we previously downloaded the CA certificate, we can also verify this certificate by using the `verify` subcommand. To use `verify`, we will give the path to the CA certificate that was previously downloaded, and the client certificate that we just downloaded, as follows:

```
$ openssl verify -CAfile ca_crt.pem mylaptop.pem
mylaptop.pem: OK
```

If your master failed to return a certificate in the previous step, use `puppet cert` on the master to find the certificate. For the `mylaptop` example, issue the following commands:

```
[root@puppet ~]# puppet cert --list mylaptop
```
If the certificate is present but unsigned, the output will have a missing + symbol at the beginning, like this:

```
[root@puppet -]# puppet cert --list mylaptop
```

If the certificate is not present, the output will look like this:

**Error: Could not find a certificate for mylaptop**

A common problem with certificates is an old certificate or a mismatch between the ca_server/master and the node. The simplest solution to this sort of problem is to remove the certificate from both machines and start again.

To remove the certificate on the ca_server, use `puppet cert clean` with the appropriate hostname, as follows:

```
[root@puppet -]# puppet cert clean mylaptop
Notice: Revoked certificate with serial 6
Notice: Removing file Puppet::SSL::Certificate mylaptop at '/var/lib/puppet/ssl/ca/signed/mylaptop.pem'
Notice: Removing file Puppet::SSL::Certificate mylaptop at '/var/lib/puppet/ssl/certs/mylaptop.pem'
```

As mentioned in the output, the certificates are stored in the subdirectories of `/var/lib/puppet/ssl`. If the `puppet cert clean` command does not remove the certificate, you can remove the files manually from this location.

On the node, remove `private_key` and `certificate` from the `/var/lib/puppet/ssl` directory manually (there is no automatic way to do this). Alternatively, you can choose to remove the entire `/var/lib/puppet/ssl` directory and have the node download the CA certificate again.

This location is different for Puppet Enterprise. Puppet Enterprise stores certificates in `/etc/puppetlabs/puppet/ssl`. This often involves less work as compared to that of finding all the files that need to be removed.

When we ran `puppet cert clean` on the master, one of the output lines mentioned that the certificate has been revoked. X509 certificates can be revoked. The list of certificates that have been revoked is kept in the **Certificate Revocation List (CRL)**, which is in the `ca_crl.pem` file in `/var/lib/puppet/ssl/ca`. 

---

[22]
We can use OpenSSL's `crl` utility to inspect the CRL, as follows:

```bash
[root@puppetca]# opensslcrl -in ca_crl.pem -text
Certificate Revocation List (CRL):
    Version 2 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: /CN=Puppet CA: puppet.example.com
    Last Update: Mar 5 06:40:51 2015 GMT
    Next Update: Mar 3 06:40:52 2020 GMT
    CRL extensions:
        X509v3 Authority Key Identifier:
    X509v3 CRL Number:
        2
    Revoked Certificates:
        Serial Number: 06
            Revocation Date: Mar 5 06:40:52 2015 GMT
    CRL entry extensions:
        X509v3 CRL Reason Code:
            Key Compromise
```

As you can see, the certificate with the serial number 6 has been marked as revoked. The serial number is located within the certificate. When the master verifies a client, it will consult the CRL to verify that the serial number is not in the CRL.


**Summary**

In this chapter, we introduced the main components of Puppet infrastructure. We highlighted the key points of a puppet agent run and the communication that takes place. We then moved on to talk about communication and how the X509 certificates are used by Puppet. We used Puppet's REST API to download and inspect certificates. In the next chapter, we will talk about Puppet manifests and how one can troubleshoot problems with code.
Where to buy this book
You can buy Troubleshooting Puppet from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.