Available as a
PDF Electronic Book
or Print On Demand

Pulling Strings with Puppet

Configuration Management Made Easy

192 PAGES

James Turnbull

Apress®
THE EXPERT'S VOICE™

Pulling Strings with Puppet

Configuration Management Made Easy

JAMES TURNBULL

Pulling Strings with Puppet: Configuration Management Made Easy

ISBN-13: 978-1-4302-0622-4

ISBN-10: 1-59059-978-0

eISBN-13: 978-1-59059-978-5

Printed and bound in the United States of America (POD)

Lead Editors: Jason Gilmore, Joseph Ottinger

Technical Reviewer: Dennis Matotek

Editorial Board: Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jason Gilmore, Kevin Goff, Jonathan Hassell, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Project Manager: Beth Christmas

Copy Ediitor: Ami Knox

Associate Production Director: Kari Brooks-Copony

Compositor: Richard Ables

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

Contents

About the Author

JAMES TURNBULL works for the National Australia Bank as a Security Architect. He is the author of Hardening Linux, which focuses on hardening Linux hosts, and Pro Nagios 2.0, which focuses on enterprise management using the Nagios open source tool.

James has previously worked as an executive manager for IT security at the Commonwealth Bank of Australia, the CIO of a medical research foundation, manager of the architecture group of an outsourcing company, and in a number of IT roles in gaming, telecommunications, and government. He is an experienced infrastructure architect with a background in Linux/Unix, AS/400, Windows, and storage systems. He has been involved in security consulting, infrastructure security design, SLA, and service definition, and has an abiding interest in security metrics and measurement.

About the Technical Reviewer

DENNIS MATOTEK was born in a small town in Victoria, Australia called Mildura. Like all small towns, the chronic lack of good strong coffee drives the young to search further afield. Dennis moved to Melbourne where good strong coffee flows through the city in a river called the Yarra. However, it was in Scotland that Dennis was introduced to Systems Administration.

Scotland, on the technological edge, had 486DX PCs and a Vax. On arriving back in Melbourne, after staying awake for 24 hours at an airport minding his bags, Dennis was given a job interview—jobs in those days fell down like snowflakes from the sky.

Since that time, Dennis has stayed predominately in Melbourne working with IBM AS400s (iSeries) for 6 years and Linux for 7 years.

Dennis also wrote and directed some short films and plays. He has a lovely LP (life partner) and a new little boy called Zigfryd whom he misses terribly when at work, which is most of the time.

Introduction

This book introduces the reader to Puppet—a Ruby-based configuration management and automation tool for Linux and Unix platforms. The book is a beginning-to-intermediate guide to Puppet. It is aimed at system administrators, operators, systems engineers, and anyone else who has to manage Linux and Unix hosts.

This book requires a basic understanding of Linux/Unix systems administration including package management, user management, using a text editor such as vi, and some basic network and service management skills. If you wish to extend Puppet, you will need to have an understanding and some aptitude with the Ruby programming language. But for simple expansion of Puppet, basic Ruby skills are all that are needed. Additionally, as a programming language, Ruby is very approachable and easy to pick up.

The book starts with explaining how Puppet works and then moves on to installation and configuration. Each succeeding chapter introduces another facet of Puppet right up to demonstrating how you can extend Puppet yourself.

Chapter 1: Introduction to Puppet

Chapter 2: Installing and configuring Puppet

Chapter 3: Puppet's configuration language

Chapter 4: Using Puppet, which you learn through practical examples

Chapter 5: Reporting with Puppet

Chapter 6: Advanced Puppet features including integration with LDAP, performance management, and scalability

Chapter 7: Extending Puppet and Facter including adding your own Facter "facts" and Puppet configuration types

All of the source code, associated scripts, and configuration examples can be downloaded from the Apress web site. You can also submit any errata at the site.

If you have any questions and queries about the book, please do not hesitate to e-mail me at `james@hardening-linux.com`.

CHAPTER 2

Installing and Running Puppet

This chapter focuses on installing and running Puppet master servers and clients (also known as nodes). There are a variety of methods you can use to install Puppet masters and clients: from source, packages, or as a Ruby Gem. This chapter will take you through the steps required for installation using each of these methods.

The Puppet server and clients are designed to run on Unix and Linux platforms; currently there is no port for Windows (although it may be possible to run Puppet under Cygwin). This book will only cover installation on Unix and Linux platforms. Both the master server and the clients will run successfully on a variety of BSD flavors, Linux distributions, Sun Solaris, Mac OS X, and indeed most Unix-like platforms that support Ruby.

The chapter will also take you through configuring and running both the Puppet master and client. By the end of this chapter, you should have an introduction to how both the master and clients can be configured. You will also be able to start and stop the master and clients on a variety of platforms.

Note ➡ When referring to the Puppet client, we'll distinguish between the terms *client* and *node*. The term *client* refers to the Puppet client daemon that connects to the Puppet master and retrieves the configuration. The term *node* refers to the underlying host to which configuration is applied.

Installation Prerequisites

The process of installing Puppet's master and client components is quick and easy, but you will need to install some prerequisites first. The prerequisites are required for both hosts that run the Puppet master or client (or both—your Puppet master can also be a Puppet node). These prerequisites include the Ruby interpreter, select Ruby libraries, and Facter.

Installing Ruby and Ruby Libraries from Packages

Many Linux distributions and Unix operating systems have Ruby packages available for them. These include Red Hat Enterprise Linux and Fedora, Debian, Ubuntu, SuSE, and Mandriva. Some distributions bundle all the required Ruby binaries and libraries in a single package. Other distributions separate the core development environment and the libraries into individual packages. In Table 2-1, I have detailed the package and/or port names for the required packages for a variety of BSD and Linux distributions.

Table 2-1. Package Names for Ruby and Ruby Libraries

OS	Ruby	Ruby Libraries	Additional Package
Debian	`ruby`	`librubylibopenssl-ruby`	`libxmlrpc-ruby`
FreeBSD	`ruby`		
Gentoo	`ruby`		
Mandriva	`ruby`		
NetBSD	`ruby`		
OpenBSD	`ruby`		
Red Hat	`ruby`	`ruby-libs`	
SuSE	`ruby`		
Ubuntu	`ruby`	`librubylibopenssl-ruby`	`libxmlrpc-ruby`

So, if we're installing Ruby and its libraries on a Red Hat Fedora host, we need to use its package management system to install the `ruby` and `ruby-libs` packages like so:

```
# yum install ruby ruby-libs
```

Installing the Ruby package and libraries may not always install all of the required libraries. If the following base libraries are not installed as part of your base Ruby installation, you may need to selectively install the missing libraries.

PlatformRubyGems	Package Name
OpenBSD	`rubygems`
Red Hat	`rubygems`
SuSE	`rubygems`
Ubuntu	`rubygems`

If there is not a RubyGems package for your platform, you can also download a source package and compile it yourself. You can find the RubyGems source package by clicking the downloads link at `http://rubygems.org/`.

Download the latest version of RubyGems and unpack it.

```
# wget http://rubyforge.org/frs/download.php/20989/rubygems-0.9.4.tgz
# tar zxf rubygems-0.9.4.tgz
# cd rubygems-0.9.4
```

We use the `ruby` binary to run the `setup.rb` script to install RubyGems like so:

```
# ruby setup.rb
```

This will install the `gem` binary, which we can check is functioning like so:

```
# gem --version
0.9.4
```

Once you have installed RubyGems, you can use the `gem` binary to install Gems such as Puppet. The Puppet Gem is located on the Reductive Labs site, and you can install it like so:

```
# gem install --remote --source http://reductivelabs.com/downloads puppet
```

Note ➡ When installing Puppet from a Gem, the Facter Gem will also be installed as a dependency.

At the end of the installation process, both the Puppet server and client will be installed.

Note ➡ You can also install the very latest cutting-edge Puppet from its Subversion source repository using the instructions you can find at `http://www.reductivelabs.com/trac/puppet/wiki/PuppetSource`.

Getting Started with Puppet

Now that we've installed Puppet, let's get the Puppet master daemon up and running and add our first node. One of the strengths of the Puppet infrastructure is that most of the functionality will run with default configuration, without any changes required on your behalf. The only two things we need to get Puppet running are a user and group to run it and a very basic configuration to apply to our first node. In this section, we will create that user and group and then look at starting the Puppet master daemon for the first time using our basic configuration.

First, we need to ensure we have a user and group for the master daemon to run as. If you've installed Puppet from a package, generally a user and group, usually both called `puppet`, will already have been created for you. You can check for this user by using the `id` command like so:

```
# id puppet
uid=503(puppet) gid=503(puppet) groups=503(puppet)
```

You could also check the `/etc/passwd` and `/etc/group` files directly:

```
# grep 'puppet' /etc/passwd
puppet:x:503:503:puppet user:/home/puppet:/bin/bash
```

If the `puppet` user and group does not exist, you need to create them. I recommend naming both user and group `puppet` as this is the default Puppet expects. So on a Red Hat host you would create them like so:

```
# groupadd puppet
# useradd -M -g puppet puppet
```

Starting the Puppet Master

If we've got a user and group to run the Puppet master server, we can start it using the `puppetmasterd` binary.

```
# puppetmasterd
Manifest /etc/puppet/manifests/site.pp must exist
```

You can see that trying to start `puppetmasterd` has resulted in an error message stating that the manifest, `/etc/puppet/manifests/site.pp`, must exist. A *manifest* is Puppet's term for a

text document that defines a particular configuration or configurations. These manifests are then compiled and applied to a Puppet node to set the desired configuration on the node.

Puppet requires a central manifest file, called the *site manifest*, before the master daemon can be started. By default, this site manifest file is called `site.pp` and is located in the `/etc/puppet/manifests` directory (you'll learn how to reconfigure this location later in this chapter). This central manifest will ultimately contain all the configuration information required to configure all your nodes, either directly in the file or by including and importing other files.

But we'll discuss your manifest configuration and how to structure it in Chapter 4. For now, we just want to create a simple `site.pp` file so we can get Puppet started. First, let's create the directory:

```
# mkdir -p /etc/puppet/manifests
```

Now, in Listing 2-1 you can see an example `site.pp` file.

Listing 2-1. Your First `site.pp` *File*

```
file { "/etc/passwd":
    owner => "root",
    group => "bin",
    mode  => 644,
}
```

This `site.pp` file is very simple: it sets the user and group ownership of the `/etc/passwd` file as well as its permissions. Indeed, our first `site.pp` file could do anything, we just need a syntactically correct file so we can start the daemon; we will add to it further and look at its syntax in Chapter 3.

Now in Listing 2-2, with our newly created site manifest, let's try to start the master daemon again.

Listing 2-2. Starting the Master Daemon

```
# puppetmasterd --verbose --no-daemonize
info: Starting server for Puppet version 0.23.0
info: Parsed manifest in 0.01 seconds
info: Listening on port 8140
notice: Starting Puppet server version 0.23.0
```

This time we've started `puppetmasterd` with the `--verbose` and `--no-daemonize` options. The `--verbose` option turns on verbose logging, and the `--no-daemonize` option forces the master daemon to run in the foreground. This mode is ideal for troubleshooting your master daemon.

Puppet expects to find each node defined in a manifest, either directly in the `site.pp` file or in another file and imported into the site manifest. The node definitions tell Puppet about each host to be configured and exactly what configuration applies to them; for example, you might have configuration specific to Debian hosts, or to web servers or hosts in a specific location. When you are using node definitions, only the configuration defined to a particular node will be applied to that node.

Puppet detects if you have any nodes defined. If you don't have any defined, as we have here, Puppet turns off node designation. With node designation turned off, all configuration resources (excluding configuration in classes and definitions, which we'll talk about in Chapter 3) defined will be applied to all nodes that connect to the master. As we don't have any nodes, nor any substantive configuration, it's easiest to turn off nodes until we're ready to define our first node. We'll look at node definition in Chapter 3.

From Listing 2-2, you can see the master daemon has started and is listening on TCP port 8140. You'll need to open this port in any firewall you have running on the local host. If the port is open and the master daemon has started without any error messages, you're now ready to connect your first node.

Starting the Puppet Client

Unlike the Puppet master daemon, the Puppet client daemon runs as the `root` user, allowing it to perform the required configuration actions on your Puppet node. The first time you start a node, it will generate a local self-signed certificate, connect to a master server (which, in addition to distributing configuration to nodes, also acts as a Certificate Authority) you specify, and request that the certificate be signed.

Tip ➡ Puppet relies on SSL to talk between client and server. You need to ensure that the time on your server and client is correct and appropriately synchronized to ensure SSL functions correctly.

Once the certificate is signed, the node will request whatever configuration is specified for that node. The master server will then compile and deliver that configuration. The configuration is then implemented on the node. The Puppet client will then periodically, by default every 30 minutes, check the master to see whether the configuration defined there is unchanged. If it has changed, the client will request a recompilation of the configuration, and the new configuration will be implemented on the node.

Tip ➡ If you're running the Puppet client on the same host as the server, your certificate will be automatically signed.

Now, let's start the Puppet client, as demonstrated in Listing 2-3.

Listing 2-3. Starting the Puppet Client

```
# puppetd --server puppetmaster.testing.com --verbose --waitforcert 60
notice: Did not receive certificate
```

We've started the Puppet client daemon with three options, `--server`, `--verbose`, and `--waitforcert`. The `--server` option tells the client the name of the server to connect to. You should specify the server in the form of a fully qualified domain name. The `--verbose` option enables verbose output for the client and stops it going into the background and daemonizing.

The last option, `--waitforcert`, tells the client to check every 60 seconds to see whether a signed certificate is returned from the server. This option is generally only used when you are connecting a new node and tells the client daemon to keep checking the server for a signed certificate. You can see in Listing 2-3 a log message indicating that the client is still waiting for the certificate from the server:

```
notice: Did not receive certificate
```

If you check on your master daemon, you can see a corresponding log message:

```
notice: Host node1.testing.com has a waiting certificate request
```

This message indicates that the client's request to have a certificate signed has been received, and now you need to act on it.

Signing Your Client Certificate

So how does our node get a signed certificate, our node authenticated, and the node configuration delivered? Certificate signing is done on the master server by the `puppetca` tool. The `puppetca` tool controls the Puppet Certificate Authority and allows certificate requests to be signed or revoked.

Note ➡ You can also configure Puppet to automatically sign all incoming certificate requests (known as *autosign*), either from every node or using coarse-grained authentication to selectively sign node requests based on hostname or domain. Using both forms of autosign poses a serious security risk as they bypass Puppet's security controls. I don't recommend using autosign. But if you do, you can see more details about autosign and Puppet's certificate management at `http://www.reductivelabs.com/trac/puppet/wiki/CertificatesAndSecurity`.

You can list all of the waiting certificate signing requests like so:

```
# puppetca --list
node1.testing.com
```

You can see the `--list` option has listed our node's signing request. Now, if we want to sign it, we can use the `puppetca` command again like so:

```
# puppetca --sign node1.testing.com
Signed node1.testing.com
```

We specify the option `--sign` together with the hostname of the node whose certificate we wish to sign, in this case `node1.testing.com`. On the next line, we can see the command has returned a message indicating that the certificate is now signed. The node is now authenticated to the server.

If we go back to the client daemon, we will see logging messages indicating that the certificate has been returned and the client has been started.

```
notice: Got signed certificate
notice: Starting Puppet client version 0.23.0
```

Then server will now compile and deliver any configuration for that node to the client daemon to be applied. In our example `site.pp` file in Listing 2-1, we're configuring the `/etc/passwd` file and have changed its group ownership, from the default of `root` to `bin`. You should now see the `/etc/passwd` file has the updated group ownership.

```
# ls -la /etc/passwd
-rw-r--r-- 1 root bin 1579 2007-08-01 19:05 /etc/passwd
```

Now you've got a simple Puppet master daemon running and have your first node connected. If you want you can now jump ahead to Chapter 3 to look at how to use Puppet to configure your hosts, or you can continue to read this chapter to learn more about how to run and configure Puppet.

Running the Puppet Daemons

Like most Unix and Linux applications, the Puppet daemons, `puppetmasterd` and `puppetd`, can be started and stopped using your platform's standard spawn process. Indeed, if you've installed Puppet from a package, you'll usually find that the package installation process has added the appropriate links and scripts to start the daemons when your host boots.

If you have manually installed Puppet from source, or your package installation has not provided a control script, you can find a variety of scripts you can use in the Puppet source package in the `conf` directory. Currently, there are scripts and configuration files for FreeBSD (which can be easily adjusted for other BSD platforms), Gentoo, Red Hat, Solaris, and SuSE. You can easily modify the files available to suit most platforms capable of running Puppet.

Tip ➡ The Puppet daemons also do some signal handling. The Puppet master and client daemons both recognize the `SIGHUP` signal, which forces the daemons to restart themselves. The `SIGINT` signal will terminate both the master and client daemons. The Puppet client also processes the `SIGUSR1` signal, which causes the daemon to initiate a new connection to the server and check for new configuration.

Configuring Puppet

Your Puppet installation comes with a number of binaries that run the various Puppet functions and daemons. We've already touched on the `puppetd`, `puppetmasterd`, and `puppetca` binaries, but we'll go into more detail on them in the sections that follow. This is not a definitive guide to every configuration option but rather focuses on the key options. For a full reference to every command-line and configuration file option, you can find a guide at `http://www.reductivelabs.com/trac/puppet/wiki/ConfigurationReference`.

Each Puppet binary can be configured via the command line or via a configuration file or files. In Table 2-6, you can see a list of all the Puppet binaries and their purposes.

Table 2-6. Puppet Binaries

Binary	Description
`puppet`	A local configuration script interpreter and executor
`puppetd`	The Puppet client daemon that runs on the managed host
`puppetmasterd`	The Puppet master daemon that manages the nodes
`puppetca`	The Puppet Certificate Authority server used to authenticate nodes to the master server
`puppetrun`	A tool that can connect to clients and force them to run their configurations
`filebucket`	A client to send files to a Puppet file bucket
`ralsh`	An interactive Puppet shell for converting current state into Puppet configuration code
`pi`	Tool to output documentation about Puppet types and providers
`puppetdoc`	Tool that prints Puppet reference documentation (generally only used within other Puppet tools)

Each binary has a different set of command-line options you can use to run and configure it. The easiest way to see the configuration options used for each binary is by executing the binary with the `--help` option like so:

```
# puppet --help
```

Note ➡ To get the `--help` text, you need to have the RDoc library installed as discussed earlier in this chapter.

Puppet configuration can also be managed via configuration file. Puppet's configuration file model is in the style of INI files. Each file is divided into namespace sections, and each section name is enclosed in parentheses and named for the Puppet function it configures; for example, the namespace used to configure the Puppet client daemon is called `[puppetd]`. The use of namespaces means options can be used in multiple namespaces, if the option is

relevant to the binary being configured. For example, you can specify the same option twice, with different values, in the `[puppetd]` and `[puppetmasterd]` namespaces, and each binary will use only the configuration option contained in its own namespace.

In Table 2-7, I've listed the key namespaces in the configuration file.

Table 2-7. Configuration File Namespaces

Section	Description
`main`	General configuration options for multiple elements of Puppet
`puppetd`	Configuration options related to the Puppet client daemon
`puppetmasterd`	Configuration options related to the Puppet master daemon

You can see an example of a Puppet configuration file in Listing 2-4.

Listing 2-4. Puppet Configuration File

```
[main]
vardir = /var/lib/puppet
logdir = /var/log/puppet

[puppetd]
localconfig = $vardir/localconfig
```

You can see we've defined two namespaces, `[main]` and `[puppetd]`, in Listing 2-4 and specified some configuration options in each. Configuration options are structured as follows:

```
option = value
```

Boolean options are structured like so:

```
option = true
```

Or:

```
option = false
```

Each Boolean option is either defined as `true` or `false`.

When parsing a configuration file, all binaries will set options contained in the `[main]` namespace and will then set any options specified in the section named for the binary being executed; for example, the `puppetd` binary will set all options in the `[puppetd]` namespace.

You can also see in Listing 2-4 that you can reuse previously defined options in other configuration options by prefixing them with `$`. For example, we defined the `vardir` option in the `main` section and then reused this value as part of the `localconfig` option in the `puppetd` section.

```
$vardir/localconfig
```

You can also use any configuration option from the Puppet configuration file on the command line by prefixing it with `--`. So to specify the `vardir` option on the command line, we would specify `--vardir` as an argument. Boolean configuration options are specified on the command line using an on/off model like so:

```
# puppetd --trace
# puppetd --no-trace
```

In the first line, the `trace` option is set on, and in the second line, it is disabled by prefixing the option with `no-`.

By default, Puppet binaries will look for their configuration in a file located in the `/etc/puppet/` directory. From version 0.23.0 of Puppet, each binary looks for a configuration file called `puppet.conf` in `/etc/puppet`. In previous versions, each Puppet binary looked for separate files, i.e., the Puppet master daemon looks for the `puppetmasterd.conf` file, the client for `puppetd.conf`, and the Puppet Certificate Authority for `puppetca.conf`.

This transition to a single configuration file is aided by the use of the `--genconfig` option. You can execute each of the Puppet binaries with this flag and a commented, default configuration file will be outputted, and the binary will exit. You can pipe this output into a file to create a configuration file like so:

```
# puppetmasterd --genconfig > /etc/puppet/puppet.conf
```

The resulting output makes an excellent starting point for an initial Puppet configuration.

In this chapter, we're going to focus on running and configuring three of the Puppet binaries; their configuration file options, `puppetmasterd`, `puppetd`, and `puppetca`; and the options contained in the general `[main]` section of the configuration file. We'll touch on the other binaries and configuration options in later chapters.

Tip ➡ You can see a full list of all configuration options and their functions at `http://www.reductivelabs.com/trac/puppet/wiki/ConfigurationReference`.

The [main] Configuration Namespace

Every Puppet binary will check the configuration file and set any configuration options found in the [main] namespace in your Puppet configuration file. These variables set the high-level options that control Puppet's environment, such as the location of the configuration directory. In Table 2-8, I've listed some of the key options you can configure in the [main] namespace.

Table 2-8. The [main] *Configuration File Section*

Option	Description
confdir	Location of the configuration directory. Calculated based on the user running the binary and defaults to /etc/puppet.
vardir	Location of dynamic data directory. Calculated based on the user running the binary and defaults to /var/puppet.
logdir	Log directory, defaults to $vardir/log.
rundir	Location of Puppet PID files, defaults to $vardir/run.
statedir	State directory, defaults to $vardir/state.
statefile	State file, defaults to $statedir/state.yaml.
ssldir	Location for Puppet's SSL certificates, defaults to $confdir/ssl.
trace	Whether to print stack traces on error, defaults to false.
filetimeout	The frequency in seconds that configuration files are checked for changes.
syslogfacility	Specifies the syslog facility to use, defaults to daemon.

The first options in Table 2-8 specify the location of a variety of Puppet resources. The confdir options tells Puppet where to look for configuration files. The value of this option is used as a default for other directory locations; for example, the default directory for SSL certificates, specified using the ssldir option, is $confdir/ssl. The default value for this option is dependent on the user that is executing Puppet. If the user is root or the user specified in the user option (in the [puppetmasterd] namespace), it defaults to /etc/puppet; otherwise, it defaults to ~.

Other directories that can be specified include the `vardir` for dynamic Puppet data and the `logdir` option that specifies the location of Puppet log files. Also configurable are the Puppet state directory and state file using the `statedir` and `statefile` options, respectively. The Puppet state directory and file hold the current state of running configuration, and the state file stores the state in YAML (a recursive acronym for YAML Ain't Markup Language—`http://yaml.org`) format.

The `trace` option turns on stack tracing for some Puppet errors. It is a Boolean option and defaults to `false`. The `filetimeout` option specifies how often in seconds Puppet will check for updates in configuration files; it defaults to 15 seconds. The last option in Table 2-8 allows you to set the syslog facility that Puppet will use. It defaults to `daemon`.

Note ➡ Puppet also has support for multiple environments, for example, production, testing, and development. There is some documentation available describing multiple environments at `http://reductivelabs.com/trac/puppet/wiki/UsingMultipleEnvironments`.

Configuring puppetmasterd

The Puppet master daemon is initiated by the `puppermasterd` binary. This is the core of the Puppet client-server model; the server compiles and provides the compiled configuration to the nodes. In this section, we'll look at some of the command-line flags and configuration file options that can be used to configure the Puppet master daemon.

There are a number of command-line flags you can pass to the binary, and you can see a list of the most useful flags in Table 2-9.

Table 2-9. `puppetmasterd` *Flags*

Flag	Description
`--daemonize` | `-D`	Daemonize the process (default).
`--no-daemonize`	Do not daemonize the process.
`--debug` | `-d`	Enable debugging (leaves process in the foreground).
`--logdest` | `-l` *file* | `console` | `syslog`	Specify logging destination (defaults to syslog).
`--mkusers`	Create the initial set of users and directories.
`--verbose` | `-v`	Enable verbose output (leaves process in the foreground).
`--help` | `-h`	Print help text.
`--version` | `-v`	Print the version.

Let's examine the flags in Table 2-9 in more detail. The `--daemonize` option tells the Puppet master daemon to daemonize the process and is the default behavior of the `puppetmasterd` binary when executed. The `--no-daemonize` option flag prevents the process being daemonized and leaving it running in the foreground. The `--debug` option causes the process to output debugging data. This is useful for troubleshooting. The `--logdest` flag lets you tell the master daemon where to output logging data; you have the choice of specifying a file name, syslog output, or the console. It defaults to syslog output.

The `--mkusers` flag only needs to be run once when you first install Puppet. It creates the required `puppet` user and group for Puppet to run as (if they haven't already been created).

Lastly, the `--verbose` option outputs all logging messages to the command line. The `--help` and `--version` options print the help text and version, respectively.

In the Puppet configuration file, there are also some useful options for the `[puppetmasterd]` namespace that you can use to configure the Puppet master daemon. You can see these options in Table 2-10.

Table 2-10. `puppetmasterd` *Namespace Options*

Option	Description
`user`	The user who should run the Puppet master daemon
`group`	The group who should run the Puppet master daemon
`manifestdir`	The directory to store configuration manifests, defaults to `$confdir/manifests`
`manifest`	The name of the site manifest file, defaults to `$manifestdir/site.pp`
`bindaddress`	The interface to which to bind the daemon
`masterport`	The port to run the Puppet master daemon on

The `user` and `group` options tell `puppetmasterd` what user and group to run as; this defaults to `puppet` in both cases. The `manifestdir` and `manifest` options specify the directory for storing manifests and the name of the site manifest file, which default to `/etc/puppet/manifests` and `/etc/puppet/manifests/site.pp`, respectively. The `bindaddress` and `masterport` options allow you to control what interface and port to bind the daemon to; these default to binding to all interfaces and to port 8140.

Configuring puppetd

The command-line operation of the Puppet client daemon is very similar to the operation of the master daemon. It can be configured both from the command line and via a configuration file, and in this section we'll look at the options that are typically specified for the daemon. In Table 2-11, you can see some of the common command-line flags you can use with `puppetd`.

Table 2-11. `puppetd` *Flags*

Flag	Description
`--daemonize` | `-D`	Daemonize the process (default).
`--no-daemonize`	Do not daemonize the process.
`--server` *name*	Name of the Puppet master server to connect to.
`--waitforcert` | `-w` *seconds*	Time in seconds between certificate signing requests.
`--onetime` | `-o`	Connect and pull down the configuration once and then exit.
`--noop`	Run in NOOP or dry-run mode.
`--disable`	Temporarily disable the Puppet client.
`--enable`	If disabled, reenable the Puppet client.
`--test` | `-t`	Enable some common testing options.
`--debug` | `-d`	Enable debugging (leaves process in the foreground).
`--verbose` | `-v`	Enable verbose output (leaves process in the foreground).
`--logdest` | `-l` *file* | `console` | `syslog`	Specify logging destination (defaults to `syslog`).
`--help` | `-h`	Print help text.
`--version` | `-v`	Print the version.

The `--daemonize` option is the default action for the `puppetd` process; if executed without options, it will run in the background as a daemon. The `--no-daemonize` option flag prevents the process being daemonized and leaving it running in the foreground. The `--server` option is used to specify the name of the Puppet master to connect to; it should be specified as a fully qualified domain name. The `--waitforcert` option only applies, as discussed in the "Starting the Puppet Client" section, for Puppet nodes without a certificate. It indicates the time in seconds in between certificate signing requests to a Puppet master. Once the node has a signed certificate, this option does nothing.

The `--onetime` option connects the client to the master, requests the node configuration, applies it, and then exits. The `--noop` option allows dry runs of configuration without actually applying the configuration. This allows you to see what new configuration will do without actually making any changes to the node. Using this with the `--verbose` option will output logging messages with the proposed changes that you can verify for correctness. On the following line, you can see an example of typical noop output:

```
notice: //File[/etc/group]/mode: is 644, should be 640 (noop)
```

You can see that the notice indicates that the `/etc/group` file's permissions are `644`, but the configuration would change that to `640`. The `(noop)` at the end of the message indicates that no change has been made.

The `--disable` and `--enable` options allow you to turn on and off the Puppet client. The `--disable` option sets a lock file that prevents the Puppet client from running. The same lock file is set by the Puppet client when running as a daemon to prevent the client from running twice. The `--enable` option removes the lock file and allows the client to run again on its normal schedule, by default checking half-hourly.

The `--test` option applies a number of common testing options including verbose logging, running in the foreground, and exits after running the configuration once (the `--onetime` option). The `--debug` and `--verbose` options enable debug and verbose output from the daemon, and the `--logdest` option allows you to specify where log data will be outputted: console, file, or syslog. The option defaults to syslog output. The last two options, `--help` and `--version`, print the help text and the version information, respectively.

There are also some options that you can specify in the configuration file to configure the `puppetd` daemon. You can see some of the available options in Table 2-12.

Table 2-12. `puppetd` *Namespace Option*

Option	Description
`server` *puppet*	The Puppet master server to connect to, defaults to `puppet`
`runinterval` *seconds*	The interval between Puppet applying configuration in seconds, defaults to 1800 seconds, or a half-hour
`puppetdlockfile` *file*	The location of the Puppet lock file
`puppetport` *port*	The port that the client daemon listens on, defaults to 8139

The `server` option is the configuration file equivalent of the command-line `--server` option and allows you to specify the Puppet master server to connect to; it defaults to puppet. The `runinterval` option controls how often configuration is applied to the Puppet

node. It is from this option that Puppet gets the default half-hourly application of configuration. The option is in seconds and defaults to `1800` seconds.

The `puppetdlockfile` option specifies the location of the lock file used by the `--disable` option to control the running of the Puppet client. The option defaults to `$statedir/puppetdlock`. The `puppetdport` option controls what port the client daemon binds to; by default this is 8139.

Configuring puppetca

The `puppetca` binary's primary purpose is to control and interact with the `puppetmasterd`'s built-in Certificate Authority. Its principal purpose, if you don't use the automatic signing of certificates (which is turned off by default), is to sign incoming requests from Puppet clients to authenticate new nodes.

Caution ➡ As discussed, autosigning of certificates is dangerous, as anyone can authenticate to your Puppet master. If you want to autosign certificates, use per-host authentication to only authenticate those hosts you identify. See `http://www.reductivelabs.com/trac/puppet/wiki/CertificatesAndSecurity` for more details.

We've already seen `puppetca`'s primary function when we connected our first node to Puppet, listing and signing the certificate requests of new nodes using the `--list` and `--sign` options.

```
# puppetca --sign node1.testing.com
```

You can specify more than one node on the command line, and you can also sign all outstanding certificate requests by specifying the `all` keyword like so:

```
# puppetca --sign all
```

You can also see some other useful command-line flags in Table 2-13.

Table 2-13. `puppetca` *Flags*

Flag	Description
`--revoke` | `-r` *host*	Revoke a node's certificate.
`--clean` | `-c` *host*	Remove a node's certificate from the master.
`--generate` | `-g` *host*	Generate a client key/certificate pair.
`--debug` | `-d`	Enable debugging (leaves process in the foreground).
`--verbose` | `-v`	Enable verbose output (leaves process in the foreground).
`--help` | `-h`	Print help text.
`--version` | `-v`	Print the version.

The `--revoke` option revokes a client's certificate. You can specify a decimal number, the certificate's hexadecimal code, or the hostname of the client node. The certificate is added to Puppet's Certificate Revocation List (CRL). You can specify the CRL using the `cacrl` option in the `puppetmasterd` namespace. The master daemon needs to be restarted to update the CRL with the revoked certificate.

The `--clean` option removes all files related to a particular node from the Puppet Certificate Authority. The option is most useful for rebuilding nodes. It removes traces of the old certificate and allows you to submit a new certificate signing request from the client.

The `--generate` option generates a certificate and key pair for the node or nodes specified on the command line.

You can also control a variety of certificate and SSL-related configuration options such as the key, the naming and location of certificates on both the master and the node, and a variety of other options. You can see these options at `http://www.reductivelabs.com/trac/puppet/wiki/ConfigurationReference`.

Tip ➡ In the future, you may also be able to have multiple master servers use a single certificate authority. At the moment this isn't fully supported, but you can read about it at `http://reductivelabs.com/trac/puppet/wiki/MultipleCertificateAuthorities`.

Resources

We've looked at installing and configuration Puppet in this chapter, and there are a number of useful resources and documentation online that can also help with this process.

You can also log tickets and bug reports at Puppet's trac site by registering at `http://reductivelabs.com/trac/puppet/register`.

Web

- *Puppet support*:

  `http://reductivelabs.com/trac/puppet/wiki/GettingHelp`

- *Puppet installation guide*:

  `http://reductivelabs.com/trac/puppet/wiki/InstallationGuide`

- *Puppet configuration reference*:

  `http://reductivelabs.com/trac/puppet/wiki/ConfigurationReference`

Mailing Lists

- *Puppet user mailing list*:

  `http://mail.madstop.com/mailman/listinfo/puppet-users`