



**INSTANT**

Short | Fast | Focused

# Puppet 3 Starter

Gain complete consistency from your systems with minimal effort  
using Instant Puppet 3 Starter

Jo Rhett

**[PACKT]**  
PUBLISHING

[www.allitebooks.com](http://www.allitebooks.com)

# Instant Puppet 3 Starter

Gain complete consistency from your systems with minimal effort using Instant Puppet 3 Starter

Jo Rhett



BIRMINGHAM - MUMBAI

# Instant Puppet 3 Starter

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: March 2013

Production Reference: 1200313

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78216-174-5

[www.packtpub.com](http://www.packtpub.com)

# Credits

**Author**

Jo Rhett

**Project Coordinator**

Sherin Padayatty

**Reviewer**

Chris Spence

**Proofreader**

Mario Cecere

**Acquisition Editor**

Akram Hussain

**Production Coordinator**

Conidon Miranda

**Commissioning Editor**

Yogesh Dalvi

**Cover Work**

Conidon Miranda

**Technical Editor**

Varun Pius Rodrigues

**Cover Image**

Conidon Miranda

**Copy Editor**

Ruta Waghmare

# About the Author

**Jo Rhett** is a DevOps engineer who has been using, promoting, and enhancing configuration management systems for over 20 years. He implemented package management functionality in CFEngine v2, before moving on to use Puppet daily for the last three years.

---

I'd like to thank Puppet Labs for continuing to support and grow the open source Puppet project. I'd also like to acknowledge the timely and extensive support provided by the diverse Puppet community through IRC and mailing list support. Puppet has become a fantastic tool through their commitment to the project.

---

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

# PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why Subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print and bookmark content
- ◆ On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.



# Table of Contents

<b>Instant Puppet 3 Starter</b>	<b>1</b>
<b>So what is Puppet 3?</b>	<b>3</b>
Building your starter server	4
Deterministic results	4
<b>Installation</b>	<b>7</b>
Step 1 – Preparing the host	7
Removing old versions of the software	7
Puppet client requirements	7
Puppet server requirements	7
Step 2 – Installing Puppet	8
Step 3 – Configuring the server	9
How Puppet authentication works	9
Step 4 – Starting up the server	10
Step 5 – Testing your first client	10
<b>Quick start – Using the core Puppet resource types</b>	<b>13</b>
Step 1 – Enabling the Puppet service	13
Step 2 – Managing software and services	14
Step 3 – Customizing one node	15
Step 4 – Synchronizing files and directories	16
Building a custom module	17
Class parameters	19
Member classes	20
Conditionals	21
Module files	21
Module templates	22
Referring to other resources	23
Setting defaults for resources	23
Notifying resources of changes	24
Controlling actions with schedules	24
Module philosophy	25



<b>Top 5 features you need to know about</b>	<b>26</b>
Reviewing system changes	26
File report store	26
Tagmail report processor	27
Puppet Dashboard	27
Custom report processors	28
Comparing and restoring files	28
Using environments to test changes	29
Testing your code	30
Release engineering	30
Running the Puppet server under Passenger	30
Using external data in your Puppet policy	32
Configuring Hieradata location	33
Creating Hieradata	33
Using Hieradata in a Puppet module	34
Learning more	35
<b>People and places you should get to know</b>	<b>36</b>
Official site	36
Community support	36
The Puppet Forge	36
Articles and tutorials	36
In-depth details	37
The Puppet language	37
Related projects	37
About Packt Publishing	39
Writing for Packt	39

# Instant Puppet 3 Starter

Welcome to the *Instant Puppet 3 Starter*. This book will provide you with all the information that you need to set up and use Puppet 3. You will learn the core components of Puppet, set up a working client and server, and start building your first custom module.

This book contains the following sections:

*So what is Puppet 3?* – In this section, we find out what Puppet does, how it works, and how it can drastically improve the consistency of your systems while reducing the effort involved in maintaining them. Learn about the difference between deterministic and procedural results. Most importantly, learn how to get better results every time by thinking about and expressing your desired outcome in a deterministic fashion.

*Installation* – In this section, learn how to download and install Puppet 3. Learn how to get your environment working immediately, without needing help from others.

*Quick start* – This section will show you how to use the core methods included within Puppet to reduce the amount of manual work and rework you do every day. Then you'll learn how to write a custom Puppet module to do something important and necessary for you. By the end of this section, you'll have Puppet working throughout your environment.

*Top 5 features you need to know about* – In this section you will learn how to perform five useful tasks for managing and expanding your Puppet ecosystem. By the end of this section, you will be able to review the history of changes made to your systems by Puppet, compare and restore the previous versions of files that were changed by Puppet, build a test environment for developing new modules, use an external data source for your configuration policies, and run the Puppet server as a Ruby on Rails application to improve performance.

*People and places you should get to know* – Puppet has a very active developer and user community. This section provides you with many useful links to the project page and mailing lists, as well as a number of helpful articles and tutorials.



## So what is Puppet 3?

In this section we will discuss how Puppet works. Most importantly, we will get straight to what Puppet can do for you in your environment. We will discuss how it can drastically improve the consistency of your systems, while reducing the amount of time you spend maintaining them.

First, let's talk about you. Yes, you sitting there, reading this. Based on the fact that you are reading this, you are likely to be responsible for managing systems. You might be a systems administrator paid to manage a diverse set of platforms without enough time to do it all. You might be a developer who is paid to write a code, but you find yourself spending too much time managing your development platform. You may be a DevOps engineer, with both sets of concerns. Or perhaps you are a modern homemaker trying to ensure that each of your children's laptops work consistently.

No matter what your objective is, there is a common theme to most situations today. There are too many details, too many parts, too many small differences between things, too much change within even the smallest environments, and not enough time to do it all. Puppet is designed for exactly these needs.

Doing more with less is exactly what Puppet is designed for. But let's go one step further; Puppet won't just save you time, it will ensure that each thing is exactly the way you want it, every time. When we are done, you will stop worrying if all the steps were followed, if every single thing was done. It will be like having a butler and a maid for your systems. You will learn to express the policy, and know that Puppet will handle all the details for you.

Let's talk a little bit about how this works. In this book you will learn to write out what I call the **Puppet policy**. This policy is built from manifests and modules. We'll talk about how to write a manifest in the very next section. We'll go on to build a custom module for managing Puppet clients. These two building blocks will comprise a policy expressing how you want things to be.

Puppet contains hundreds of built-in resource types that can handle many common systems' management tasks. In many situations, you won't have to write any more specific code. You express what you want in the policy using the built-in types, and Puppet handles the details for you.

However, you may have needs that are custom and unique to your environment. Puppet is very extensible and you'll find it very easy to build your own modules. In these modules, you can define your own resources to do things that the Puppet maintainers never dreamed about. We're going to show you how to do this in the *Building a custom module* section.

There are many benefits of using something which is easy to extend. One of them is that many people extend the software to meet their own needs. The other is that these people can form a community of users who share the modules with each other. So before you go off to build a module yourself, you should check out the **Puppet Forge**, a site containing thousands of modules and custom resources that others have already developed.

To summarize, we are going to show you how Puppet can help you do more, do it faster, and spend less time worrying about it. We are going to teach you how to extend Puppet to meet your specific needs. And we are going to introduce you to the large active community of Puppet users and developers who share their own custom modules in the Puppet Forge.

Enough with the talking. We are all busy people, it is time we got started!

## Building your starter server

You can run your puppet servers and clients on many different platforms and architectures. To get up to speed as quickly as possible, we recommend that you use Red Hat Enterprise Linux 6 or CentOS 6 for your server. There are good reasons for this, which can be stated as follows:

- ◆ Puppet Labs supports Red Hat and CentOS very well. Many of their supplied modules only support the RHEL/CentOS systems. So you won't run into version or compatibility issues while you are learning to use Puppet.
- ◆ All the software you need is available in binary packages for these platforms.
- ◆ The binary packages include the `init` scripts and default configurations that you may otherwise have to create manually.

After you have finished this book, you can migrate the server to any platform you wish to support. This guide will help you set up the server in a manner which makes it trivial to move it to another host server.

## Deterministic results

I am going to talk about the difference between deterministic results and procedural results. I will explain how Puppet is oriented around deterministic results, and why the best results can be achieved by learning to think in a deterministic manner.

What are deterministic results? Determinism is something that returns the same result every time, given the same input. A calculator or spreadsheet both return deterministic results. You might be thinking to yourself, "isn't all software deterministic?" Not in the sense that we are talking about. Most software applications don't return the same results every time; you have to check the environment in which the software runs, and modify the input you give it. For example, let's use a very simple program that creates a new user:

```
$ useradd -u 2000 deterministic
```

That worked fine, didn't it? Now let's run the same program again, with the exact same input.

```
useradd: user 'deterministic' already exists
```

Ah, same input but not the same result; that's not deterministic at all! To actually make this very simple idea deterministic requires a larger script that validates each value against the environment:

```
#!/bin/sh
USER=$1
USERID=$2
FOUND=$(id -u $USER 2> /dev/null)
if [ $? -ne 0 ]; then
    useradd -u 2000 deterministic
else
    if [ $FOUND -ne $USERID ]; then
        usermod -u 2000 deterministic
    fi
fi
```

This is how most programming is done. You have to check A, then B, and then do C or D based on what you observe. This is called **Procedural programming**. Think about how much more code would be required if you wanted to ensure that the gid, the comment, the home directory, and the password were correct? What about the extra code to make this function work properly for adding a root-equivalent user with UID 0? Do you agree that this would be quite a long script? You probably have a few scripts like this around. In all likelihood you end up fixing them every time you get a new platform to support, or a new version of the operating system.

Now, let's look at how you would approach this with Puppet. In Puppet, you express the ideas in a deterministic manner. By this, I mean to say that instead of expressing the steps to get to a result, you only express what you desire the final result to be. For our previous simple example, that could be as easy as this:

```
user { 'deterministic': uid => 2000 }
```

If you want it to be much more specific, you can ensure that all the parameters for this user are defined, with less lines of text than our previous simple script:

```
user { 'deterministic':
    ensure => present,
    uid    => 2000,
    gid    => 2000,
    comment => 'Deterministic User',
    home   => '/home/determinism',
    shell  => '/bin/bash',
}
```

Instead of spending time to write out A then B then C or D, you simply express the final result that you desire. In Puppet this is called a **policy**. You'll find that expressing your needs with a deterministic policy (definition of a final result) saves a lot of time, especially if you manage a heterogeneous environment, where the commands used to check for or create a user differ. You don't need to think about those differences when you are writing Puppet policies.

I'd like you to stop and reread this section one more time before we proceed. Here we have identified a very important shift in how to approach systems management. As the new users start developing their first puppet policies, it is common to see them fall back to the procedural thinking styles. They can do things with a procedural style in Puppet, but they will be fighting the current, going uphill, and wasting time. After a short while, they realize the utility of the deterministic manner and start defining their policy accordingly. They learn to do more with less, and do it faster than ever before.

I am going to try and help you avoid that first round of mistakes. We are, together, going to get you working like an experienced hand at Puppet by the end of this short book. The first step in avoiding that introductory round of wasting time is to go back and reread this section. Come to terms with the simplicity of expressing a final result.

## Installation

In this section we will walk you through the installation of the Puppet server and agent software. Then we will discuss the security model used by Puppet and how to configure the Puppet server to best avoid problems in the future. Finally, we will cover how to connect and authorize your first agent (client).

### Step 1 – Preparing the host

Before you install Puppet, you will need to check that you have all of the required elements, listed as follows:

#### Removing old versions of the software

Before we start this process, ensure that no previous versions of Puppet, Facter, Ruby, or RubyGems are installed:

```
$ sudo rpm -e puppet* facter ruby rubygems
```

#### Puppet client requirements

Following are the system requirements for the Puppet client:

- ◆ Disk space: 10 MB free
- ◆ Memory: 256 MB minimum
- ◆ Puppet V3 requires Ruby 1.8.7 or 1.9.3+ (If you are using an older version of Linux, the Puppet Labs repositories we are using provide newer versions of Ruby)

#### Puppet server requirements

Following are the system requirements for the Puppet server:

- ◆ If you use File storage for Puppet reports, you will need at least 2 MB per day to store the Puppet change history for each client system running Puppet. So if you want to keep 60 days of reports on hand for ten systems, you will require at least 1.2 GB of space.
- ◆ Puppet comes with its own server, but large installations will need Phusion Passenger or NGINX to handle many concurrent connections.



## Step 2 – Installing Puppet

The best way to install Puppet is to use the packaged builds provided by Puppet Labs. If you are using a system listed on the following web page, installing Puppet will be quite easy:

[http://docs.puppetlabs.com/guides/puppetlabs\\_package\\_repositories.html](http://docs.puppetlabs.com/guides/puppetlabs_package_repositories.html)

If you are using CentOS or Red Hat, installation could be as easy as the following:

```
$ sudo rpm -i http://yum.puppetlabs.com/el/6/products/x86_64/puppetlabs-release-6-6.noarch.rpm
```

```
$ sudo yum install puppet puppet-server
```

If you are using CentOS 5, just change the path to /el/5 and your architecture. The Puppet Labs repository includes Ruby 1.8.7, so you won't have to do anything special to install puppet.

Debian is likewise simple:

```
$ wget http://apt.puppetlabs.com/puppetlabs-release-precise.deb
```

```
$ sudo dpkg -I puppetlabs-release-precise.deb
```

```
$ sudo apt-get update
```

```
$ sudo apt-get puppet
```

Windows packages can be downloaded from <http://downloads.puppetlabs.com/windows/>.

Even if you prefer to build things from source, I recommend that you use the packages provided by Puppet Labs for this starter. If you have a platform that is not listed on the Puppet Labs repositories page or can't use the prebuilt packages, you can install Puppet from Ruby gems:

```
$ sudo gem install puppet
```

Or directly from source:

```
$ wget http://downloads.puppetlabs.com/puppet/puppet-latest.tgz
```

```
$ tar xvzf puppet-latest.tgz
```

```
$ cd puppet
```

```
$ sudo ruby install.rb
```

Both of these installation methods require many manual steps, which will be unique to your platform and beyond the scope of this book. You will need to follow the steps documented at the following link:

<http://docs.puppetlabs.com/guides/installation.html>

## Step 3 – Configuring the server

At this point you have installed Puppet and the Puppet server packages. Right now, before you start the server up, we're going to save you a lot of grief and a few trips to the mailing list. The single most common query on the list deals with confusion over how Puppet's mutual authentication works. Let's nip this right away, and save you the trouble:

1. Open `/etc/puppet/puppet.conf` in your favorite editor.
2. Find or create a section starting with `[master]`.
3. Edit or create the following lines:

```
[master]
certname = puppet.example.net
vardir = /var/lib/puppet-server
```

You should make `certname` be a name inside your own domain, and `vardir` can be any directory you like. However, I recommend that you do not use the name of a real host, or the standard `var` client directory (`/var/lib/puppet` on most platforms). Use an alias for the server right now. These two settings will allow you to move your Puppet server to a new host without any difficulty later on.

I'll explain why after you add this new hostname to your DNS. Go ahead, I'll wait.

### How Puppet authentication works

Now that you're back, we are going to talk about how Puppet handles authentication. Puppet uses SSL certificates to authenticate hosts. Each puppet client creates a certificate request and submits it to the server. The server signs each request with the `certname` mentioned earlier. This creates a closed system containing an authorized public/private key pair for each client. If you don't understand what I just said, don't worry about it; this will all become very obvious and easy to use in the next step.

So why is setting `certname` so important? The `certname` is the name of the **Puppet Certificate Authority**, which signs every client certificate. If `certname` ever changes, you have to delete and recreate every client certificate.

Time after time, on the mailing list, we see someone build a test instance and get a client going. They proceed to the next client that connects to the server using a different name, and it doesn't work. Or they move their test instance to another server, and all of their clients stop working. This is because the certificate name no longer matches the server name.



Set `certname` in advance to something abstract, and you can move your server around without any authentication problems in the future.

## Step 4 – Starting up the server

At this point we are ready to start up the server. The following command works on almost all platforms:

```
$sudo /sbin/service puppetmaster start
```

You will get the following response:

```
Starting puppetmaster: [OK]
```

Check out `/var/log/messages` to ensure no errors happened. The logs will show that it created a new certificate authority based on the name you gave it earlier.

If you ever need to start again from scratch, you can always stop the server, remove the directory named in `vardir`, and restart the server to get a fresh, clean start:

```
$ sudo /sbin/service puppetmaster stop
Stopping puppetmaster: [OK]
$ sudo rm -rf /var/lib/puppet-server
$ sudo /sbin/service puppetmaster start
Starting puppetmaster: [OK]
```



The highlighted lines in the preceding code snippet are the displayed output when you enter the command.

## Step 5 – Testing your first client

Now let's configure the first client, the system running the puppet server. You will see the system authenticate by its real hostname, not the puppet alias we created in step 3. This makes it easy if you move the puppet server to another host later.

1. If not done previously, install the puppet package with `yum install puppet`.
2. Open `/etc/puppet/puppet.conf` in your favorite editor.
3. Find or create a section starting with `[agent]`.
4. Edit or create the following lines using the same name as we used in Step 3:

```
[agent]
server = puppet.example.net
```

Now we create a relationship with the Puppet server. This involves a handshake on both sides, so you will need two logins, one for the client and one for the server.

On the client, run the following command:

```
$ sudo puppet agent --test
```

You'll see that the client creates a certificate request and attempts to connect to the server:

```
Info: Creating a new SSL key for myhost.example.net
Info: Caching certificate for ca
Info: Creating a new SSL certificate request for myhost.example.net
Info: Certificate Request fingerprint (SHA256): 12:34:56:78:9A:BC:DE...
Exiting; no certificate found and waitforcert is disabled.
```

The final sentence isn't an error. This gives us time to sign the certificate request on the server. On the server, use the following command to see the authentication requests:

```
$ sudo puppet cert list
```

You will get the following response:

```
"myhost.example.net" (SHA256) 12:34:56:78:9A:BC:DE...
```

Look at the certificate request from the client. Compare the fingerprint and then sign the certificate as follows:

```
$ sudo puppet cert sign myhost.example.net
```

You will get the following response:

```
Notice: Signed certificate request for myhost.example.net
Notice: Removing file Puppet::SSL::CertificateRequestmyhost.example.net
at '/var/lib/puppet-server/ssl/ca/requests/myhost.example.net.pem'
```

And now in the final step, we will connect with the client again. It will download the signed certificate and complete its first transaction with the server:

```
$ sudo puppet agent --test
```

You will get the following response:

```
Info: Caching certificate for myhost.example.net
Info: Caching certificate_revocation_list for ca
Info: Retrieving plugin
Info: Caching catalog for myhost.example.net
Info: Applying configuration version '1357512658'
Info: Creating state file /var/lib/puppet/state/state.yaml
Notice: Finished catalog run in 0.06 seconds
```

At this point you have seen the client establish a connection with the server and download the node's catalog. We will explain what the catalog is in the next section. There's nothing in the policy just yet, so there is nothing in the catalog. But the secure relationship between the client and the server has been created and we'll be using this to test our policies.



The server retains a copy of the client certificate. The client keeps a copy of the server's certificate. During the connection process, each uses that stored information to authenticate the other. Neither name can change without breaking the mutual authentication process.

You now have a working installation of Puppet. Let's move to the next section and create a basic policy.

## Quick start – Using the core Puppet resource types

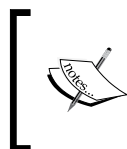
This section will show you how to use the core resource types included within Puppet to reduce the amount of manual work and rework you do every day. By the end of this section, you'll have puppet maintaining some services in your system.

### Step 1 – Enabling the Puppet service

At this point in the process, it would be essential for me to tell you how to ensure that Puppet runs after reboot. It would be `chkconfig` on the RHEL systems, or `update-rc.d` on Debian, or `smf` on Solaris. But hey, why don't we use Puppet itself and let you see how Puppet saves you time and effort?

Use your favorite editor to edit `/etc/puppet/manifests/site.pp`. Add the following code:

```
node default {
  service { 'puppet':
    ensure => running,
    enable => true,
  }
}
```



`/etc/puppet/manifests/site.pp` is the one and only filename known by the Puppet server. All other files will be sourced or named from this file. I refer to this file as the origin of the Puppet policy.

`ensure => running` means that the Puppet agent daemon should be running. This might sound like a catch-22, but it's not really. You can run the Puppet agent manually (we will do this throughout the book.) This service resource ensures that the daemon is running, and starts it if it is not.

`enable => true` ensures that the process is configured to start up at boot time. This means different things on different platforms, but Puppet allows us to not focus on those details.

Now go to a client node and run the following command. You will see it apply this policy to the system:

```
$ sudo puppet agent --test
```

You will get the following response:

```
Info: Retrieving plugin
Info: Caching catalog for myhost.example.net
Info: Applying configuration version '1357518358'
Notice: /Stage[main]/Node[default]/Service[puppet]/ensure: ensure
changed 'stopped' to 'running'
Notice: Finished catalog run in 0.95 seconds
```

If you want to stop the Puppet daemon, you could easily change the policy to the following code. During the next Puppet run it will disable the service and stop the daemon:

```
service { 'puppet':
  ensure => stopped,
  enable => false,
}
```



Let's stop for a moment and talk about the **catalog**, which is mentioned in the second line of preceding output. The catalog is a distilled, compiled version of the Puppet policy which contains only those directives necessary for the client node.


Obviously our previously mentioned test policy would apply to every node, so they would receive very similar catalogs. We will show you how to differentiate the policy given to each node in step 3.

## Step 2 – Managing software and services

You can't start services if their packages aren't installed. So why don't we tweak the default node and add a little bit more code? The following segment ensures that the Puppet package is installed, and is the latest version. If the Puppet package is updated, the Puppet service will restart itself to get the new version:

```
package { 'puppet':
  ensure  => latest,
  notify  => Service['puppet'],
}
service { 'puppet':
  ensure  => running,
  enable  => true,
  subscribe => Package['puppet'],
}
```

Notice how easy it is to set up dependencies between items? `notify` and `subscribe` are actually redundant here. You only need one or the other to build the linkage between these two resources. But if you're a belt-and-suspenders kind of person, it never hurts to list the dependency in both the places.


 We have introduced another important concept here. By default, declarations in the Puppet policy can happen in any order. You use statements like `require`, `before`, `notify`, and `subscribe` to define dependencies that ensure that resources are applied in the appropriate order. We will cover ordering and dependencies in the next section.

### Step 3 – Customizing one node

Now let's go a little further and customize the policy for one node. In the following section, we are going to ensure that the Puppet server is running the latest version of the Puppet master. Just below the default node block, let's add the following code:

```
node puppet-server-name inherits default {
  package { 'puppet-server':
    ensure => latest,
    notify => Service['puppetmaster'],
  }
  service { 'puppetmaster':
    ensure => running,
    enable => true,
    subscribe => Package['puppet-server'],
  }
}
```

The `inherits` syntax of the node line says "do everything most nodes do, and also include this policy". You could leave off `inherits default` and the node would only execute the policy specifically within the node block. In this step we have now customized the `catalog`, which will be given to this one node.

 The name you put in the node block is not the `certname` from the previous section, but the hostname of the box where the Puppet server is running. The Puppet server is really just the policy keeper. Policies are applied by the Puppet agent, even on the node which is running the server.



## Step 4 – Synchronizing files and directories

Let's enable file synchronization down to the client node. First, we need to enable the file server.

Use your favorite editor to edit `/etc/puppet/fileserver.conf`:

```
[files]
  path /etc/puppet/files
  allow *
```

Now let's create some test files for this example. We'll make the directory owned by you so that we don't have to sudo every command:

```
$ sudo mkdir /etc/puppet/files
$ sudo chown `id -u` /etc/puppet/files
$ cp /etc/hosts /etc/puppet/files/
$ cp /etc/motd /etc/puppet/files/
```

Now let's put some file resources here as examples. You can put the following policy text in the default node block (for all hosts), or you can put it inside a node block for a single host:

```
file { ['/tmp/hosts']:
  ensure => file,
  owner  => nobody,
  group  => nobody,
  mode   => 0444,
  force  => false,
  source => 'puppet:///files/hosts',
}

file { ['/tmp/hosts.linked']:
  ensure => link,
  target => '/tmp/hosts',
}

file { ['/tmp/puppet-files']:
  ensure => directory,
  owner  => root,
  group  => root,
  mode   => 0444,
  recurse => true,
  source => 'puppet:///files',
}
```

Now let's test out the revised policy on your node:

```
$ sudo puppet agent --test
```

You will get the following response:

```
Info: Retrieving plugin
Info: Caching catalog for myhost.example.com
Info: Applying configuration version '1357520339'
Notice: /Stage[main]/Node[myhost]/File[/tmp/hosts]/ensure: defined
content as '{md5}41e6824ef17c17aa577cd6fd6f794351'
Notice: /Stage[main]/Node[myhost]/File[/tmp/hosts.linked]/ensure:
created
Notice: /Stage[main]/Node[myhost]/File[/tmp/puppet-files]/ensure:
created
Notice: /File[/tmp/puppet-files/hosts]/ensure: defined content as '{md5}4
1e6824ef17c17aa577cd6fd6f794351'
Notice: /File[/tmp/puppet-files/motd]/ensure: defined content as '{md5}
d41d8cd98f00b204e9800998ecf8427e'
Notice: Finished catalog run in 0.51 seconds
```

With a simple `ls -la /tmp` you'll be able to see the file, link, and directory that this policy has created.

At this point we've created a small but very effective Puppet policy. You have three different ways to copy files down to hosts. Packages are installed, and services are stopped or started based on your policy. And yet, the entire policy as expressed in the `site.pp` file, still fits on a single page. In this you can start to perceive the power of Puppet.

## Building a custom module

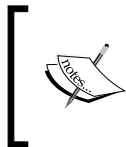
Modules are self-contained classes that provide new resources, facts, types, providers, and functions for use in your puppet policy. Each module has its own namespace, so its variables will not conflict with the variables within your policy, or within another module. You can create your own modules, or you can download them from the Puppet Forge.

Module names should only contain *lowercase letters, numbers, and underscores*, and should begin with a lowercase letter. There is a reason for the last requirement, which I will explain later.

Modules are installed in your module path, usually `/etc/puppet/modules`.

Now we shall take the examples from our previous section, and build a complete module for maintaining the puppet configuration on your clients and the server. This idea could be easily expanded to maintain any other application. The first step is to build the correct directory structure for a module. The following puppet command will create all of the necessary files and directories for a module to operate as expected:

```
$ cd /etc/puppet/modules
$ puppet module generate my-puppet
Notice: Generating module at /etc/puppet/modules/my-puppet
my-puppet
my-puppet/spec
my-puppet/spec/spec_helper.rb
my-puppet/README
my-puppet/manifests
my-puppet/manifests/init.pp
my-puppet/tests
my-puppet/tests/init.pp
my-puppet/Modulefile
$ mv my-puppet puppet  ##(see tip below)##
$ cd puppet
$ mkdir files templates
$ cd manifests
```



For reasons unclear to me, the Puppet module will only generate modules with a dash in the name, but dashes are not legal in module names used in your policy. So we rename the module immediately after creating it.

The one file that must exist in every module is `manifests/init.pp`. This file must contain a class with the same name as the module. Use your favorite editor and open that file now. Set up the initial contents as follows:

```

class puppet(
  $version      = 'latest',
  $status       = 'running',
  $environment  = 'production',
  $server       = 'puppet.example.net',
) {
  package { ['puppet':
    ensure => $version,
    notify => Service['puppet']
  ] }
  service { ['puppet':
    ensure => ${status},
    enable => true,
    require => Package['puppet'],
  ] }
}

```

You will observe that this is much like the Puppet policy we used in `site.pp`. You will also note that we have supplied four new variables but only used one of them. We will get to that shortly. For now, let's go back to `site.pp` and remove this policy. In `site.pp`, we will replace that with the following invocation of our module:

```

node default {
  class { ['puppet': ] }
}

```

Much shorter and easier to read, isn't it? We have replaced a block of code with a simple module invocation. This is the main benefit of modules; they allow you to make extensive changes with small and easy-to-read policy statements.

## Class parameters

Now, what were those variables we declared before? Those are called class parameters. They define the information that is required for the class to operate. You will notice that in our example, all four parameters were given values. This allows someone to use the class without supplying any information. They will receive the default values that we specify here. If we did not supply a default value, the parameter would be required, and invoking the class without the parameter would create an error.

To supply a parameter in your policy, just add it to the class invocation as follows:

```

node default {
  class { ['puppet':
    version => '3.2.0'
  ] }
}

```

## Member classes

Within a module you can create other classes. These are the supporting classes which make up the module. You can use them within the main module class, or you can invoke them directly in the policy. We shall create a member class now, which we shall use for a more specialized configuration. Let's take the remaining Puppet policy out of `site.pp`, and add it to our `server.pp` file here:

```
class puppet::server(  
  $version = 'latest',  
  $status  = 'running',  
  $onboot  = true,  
) {  
  package { 'puppet-server':  
    ensure => $version,  
    notify => Service['puppetmaster'],  
  }  
  service { 'puppetmaster':  
    ensure => $status,  
    enable => $onboot,  
  }  
}
```

Now, go back to your `site.pp` policy file and change the block for the node running the Puppet server to the following:

```
node puppet-server-name inherits default {  
  class { 'puppet::server':  
    version => '3.2.0',  
  }  
}
```

We have now created a member class that handles the needs of hosts, which are running the Puppet master service. Every host will be assigned the `puppet` class. Only our server will be assigned the `puppet::server` class. This type of opt-in logic is much easier to read and maintain than the `if/then/else` structures based on the hostname.

## Conditionals

If you support different operating systems, you may find that some values need to differ from system to system. For example, you may find that the package name for Puppet in Red Hat-derived Linux is `puppet`, but on FreeBSD you need `puppet3`. In this case, you would check `fact` (covered in the next section) to differentiate between the two operating systems. Here is how to use a selector to handle that situation:

```
$packagename = $operatingsystem ? {
  'redhat'   => 'puppet',
  'freebsd'  => 'puppet3',
  default    => 'puppet',
}
package { ${packagename}:
  ensure => $version,
  notify => Service['puppet'],
}
```

Puppet also supports the `if/else` syntax. However, in all the years I've been working with puppet, I have come to recognize that very few situations demand the `if` statements. An `if` statement inside a module hides a policy decision in the guts of a module. For most of the time, you should be using the member classes as documented before, so that the policy is clearly spelled out in the site manifest.

## Module files

One of the things that a module might contain is files or directories, which should be distributed to the clients. In fact, would this not be a great way to synchronize the Puppet configuration on each host? First, let's copy the `puppet.conf` file we created during the installation into the module's files directory:

```
$ cd /etc/puppet/modules/puppet/
$ cp /etc/puppet/puppet.conf files/ puppet.conf
```

Now, let's add some code to the Puppet class to keep this file up-to-date on all systems:

```
file { ['/etc/puppet/puppet.conf':
  ensure => file,
  source => 'puppet:///modules/puppet/puppet.conf',
  owner  => root,
  group  => root,
  mode   => 0444,
  force  => true,
  notify => Service['puppet'],
}
```

Now every client will receive exactly the same `puppet.conf` file. Any changes you make to this file will be distributed to every host that is assigned to the Puppet class.

## Module templates

What if you need to have some differences in the configuration on each host? And what are we going to do with those parameters at the top of the class? Templates are the answer to both of those questions. We will use those parameters to build a custom Puppet configuration based entirely on class parameters.

The first thing we will do is copy the Puppet configuration file to the `templates` directory:

```
$ cd /etc/puppet/modules/puppet/  
$ cp /etc/puppet/puppet.conf templates/puppet.conf.erb
```

Next, we shall edit the file to use the variables defined at the top of the class. Open this file in your favorite editor and add or modify the following lines in the `[agent]` section:

```
server      = <%= @server =>  
environment = <%= @environment =>
```

Finally, let's modify the code to the Puppet class to build the contents of this file from the template:

```
file { '/etc/puppet/puppet.conf':  
  ensure => file,  
  content => template('puppet/puppet.conf.erb'),  
  owner  => root,  
  group  => root,  
  mode   => 0444,  
  force  => true,  
  notify => Service['puppet'],  
}
```

You have now defined a custom template, which can be edited entirely by statements in the site manifest as follows:

```
node /testlab/ {  
  class { 'puppet':  
    server      => 'labhost.example.net',  
    environment => 'staging',  
  }  
}
```

## Referring to other resources

You have probably noticed from the preceding examples that in most situations the Puppet resources are lowercase, but in some cases the resource types are capitalized. Why is that? When should you use upper or lowercase for the resource types?

When you define new resource, you must use lowercase:

```
service { 'puppet: ensure => running }
```

When you later refer to that specific resource, you capitalize the resource type. Think of this as a proper noun. In the preceding example you have created a service. After you have created and named this service, you use the capitalized form to refer to it:

```
package { 'puppet:
  ensure => present,
  notify => Service['puppet'],
}
```

A lowercase resource declares a specific instance. A capitalized resource type refers to an existing resource defined somewhere else. Okay, the proper name analogy isn't quite accurate because we capitalize the resource type, not the resource title. However, it remains the easiest way to think of it.

## Setting defaults for resources

You will also use uppercase when you want to define some default attributes for a resource. In this situation you use the capitalized resource type without a title. Here are some examples of defaults for some resources:

```
User {
  managehome => true,
  shell      => '/bin/bash',
}
Package {
  schedule => 'daily',
}
```

With these definitions, all the packages in a module will be checked in the daily schedule, unless explicitly defined otherwise, and all users will have the Bash shell unless explicitly overridden. This can save you a great deal of redundant lines in some circumstances.



You can define defaults at a global level in `site.pp`, and then override them within a module if you desire.



## Notifying resources of changes

You may want to limit how often a puppet resource is applied. For example, the Puppet `Exec` resource type allows you to run an arbitrary command. Unfortunately, the `Exec` resource will run during every Puppet run. There are two ways to limit how often a resource is applied: run it only when notified, or limit the runs by a schedule. Here are some examples:

You can tell a resource to only apply itself to the system when another resource notifies it. This is done by defining the `refreshonly` attribute. Here's an example that updates a file containing a list of installed packages every time a new package is installed:

```
# This defines the default, so all packages will notify the exec
Package {
  notify => Exec['save-package-list'],
}
exec { 'save-package-list':
  command      => 'rpm -qa > /var/tmp/packages-installed.txt',
  refreshonly  => true,
}
```

Since a resource is only applied once, the command will be run only once no matter how many packages were installed during the Puppet run. If we did not have the `refreshonly` attribute, the command would be run every time the Puppet agent ran.

## Controlling actions with schedules

Another way to limit how often or when a resource is applied is by using a `schedule`. A `schedule` allows you to specify the hours a resource may be applied within, and how often within a given period. For example, I prefer to limit the package upgrades to early in the day when people will be around to deal with any issues that may arise:

```
# Schedule in the early part of the working day (not peak)
schedule { 'early-day':
  range      => '9 - 13',
  period     => daily,
  periodmatch => number,
}# default for all packages
Package {
  schedule => 'early-day',
}
```

You can read more about defining schedules at <http://docs.puppetlabs.com/references/latest/type.html#schedule> and more about the `schedule` and `refreshonly` metaparameters that you can apply to any resource at <http://docs.puppetlabs.com/references/latest/metaparameter.html>.

## Module philosophy

If you step back and look at how your site policy `site.pp` has changed, you will notice that we have done something very important. We have moved all of the nitty-gritty details (such as the exact name of the Puppet package on each platform) out of the site policy, while leaving all the control and characterization there.

A well-built module hides all of the detailed work of implementation, without specifying the policy. The site manifest defines whether to install a module, what version it should be installing, and whether it should be enabled. This makes it to use the site manifest as a documentation for the site configuration. It becomes easy to expand a configuration to include a whole new class of hosts without any changes the modules.

Likewise, you can add an entirely new operating system to your environment without changing your site policy. In that situation you would edit the modules as necessary, with selectors for each distinct characteristic of the new operating system.

The site manifest expresses what you want. The modules are like butlers and maids; components which implement policy without bothering you with the details. You will find that this approach enables you to do more, faster, and easier than ever before.

## Top 5 features you need to know about

In this section we will discuss five ways to extend and enhance Puppet in your environment. We'll show you how to review the changes Puppet has made, get a list of backups of files, use environments to test the changes, improve the Puppet server's performance, and use data from other applications in the Puppet policy.

### Reviewing system changes

The easiest way to review the history of changes made to the system is to examine the Puppet log. This is normally in the `messages` log from `syslog`, but you can also enable direct file logging in the `puppet.conf` configuration. If you are just looking for the recent changes on a client system, the following is likely all you need:

```
$ grep puppet-agent /var/log/messages
```

You can also have the Puppet agent send a report to the server. The server can store or process these reports. Let's see how to set that up now. The first step is to enable the sending of reports to the server. Add the following line to `/etc/puppet/puppet.conf`:

```
[agent]
  report = true
```

Now we need to tell the server what to do with the reports. Here is an example setting for the server:

```
[master]
  reports = tagmail, store
```

This tells the server to store the Puppet report and to give it to a report processor named `tagmail`. You can also create your own report processor and list it here.

### File report store

If you enable the `store` method, the puppet reports from each client node will be stored in a YAML format on the disk. Assuming that you used the paths recommended in this book, the directory `/var/lib/puppet-server/reports/hostname` will contain one YAML file for each Puppet run. These will build up over time. If you enable this, you should also create a cron job to remove the older reports. Here's an example that Puppet can enforce:

```
cron { 'clean-up-reports':
  ensure => present,
  command => 'find /var/lib/puppet-server/reports -type f -name
  \*.yaml -mtime +21 -delete',
  user    => puppet,
```

```

weekday => 0,
hour    => 2,
minute  => 11,
}

```

## Tagmail report processor

It is useful when starting to send reports to the `tagmail` report processor. This allows you to send messages about changes to systems to people in e-mail. The message content can be based on either the log level of the message, or the tag applied to the resource. Here are some examples that I find useful. You define which messages should go to which e-mail addresses in the `/etc/puppet/tagmail.conf` file. Following are some useful examples:

```

# Send every log message at every level to me
all: me@example.net

# Send critical messages to my pager
alert,emergency,crit: 4155551212@vtext.com

# Send log messages from the apache module to the webmaster
apache,!info,!debug: webmaster@example.net

# Send every change notification or error to the operations team
notice,warning,err,alert,emergency,crit: operations@example.net

```



The syntax may remind you of `syslog.conf`; however, log levels are not inclusive of higher levels. If you put just `info` as a tag, you will receive `info` messages, but nothing from `notice`, `alert`, `critical`, and so on.

Tagmail is documented at

<http://docs.puppetlabs.com/references/latest/report.html#tagmail>.

## Puppet Dashboard

**Puppet Dashboard** is a product provided by Puppet Labs to store reports in a database and provides a web UI to review those changes. After you have installed and configured the Puppet Dashboard, you can enable it as a report processor in `/etc/puppet/puppet.conf`:

```

[master]
reports = tagmail, store, http
reporturl = http://dashboard.example.net/reports/upload

```

At the time of printing of this book, the Puppet Labs documentation for installing Dashboard looks very intimidating. If you are using RHEL 6 or CentOS 6 as we suggested with the Puppet Labs repositories, you can safely ignore the *Installing Dependancies* section of the installation and just install the dashboard as follows:

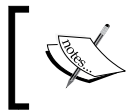
```
$ sudo yum install mysql-server puppet-dashboard
```

Then perform the configuration steps documented at <http://docs.puppetlabs.com/dashboard/manual/1.2/bootstrapping.html#configuring-dashboard>.

At the time of printing, the page doesn't mention that the Puppet Labs-provided packages include the appropriate startup scripts for you. You don't need to create your own as the page suggests:

```
$ sudo /sbin/service puppet-dashboard start
$ sudo /sbin/service puppet-dashboard-workers start
$ sudo /sbin/chkconfig puppet-dashboard on
$ sudo /sbin/chkconfig puppet-dashboard-workers on
```

Puppet Dashboard won't work very well unless you run it under Passenger. This is well documented in the manual. We will discuss this in the Passenger section.



The documentation expects you to run Dashboard on TCP port 3000. I find it easier to run Puppet Dashboard on port 80, since the users will be connecting to it with their browsers.

## Custom report processors

You can build your own custom report processors to do anything you want with the provided data. The guidelines for writing your own custom report processors are available at <http://docs.puppetlabs.com/guides/reporting.html>.

## Comparing and restoring files

When Puppet changes a file, it makes a backup of the file and stores it on the disk in the directory specified by the `clientbucket` configuration option. You can pull the old versions of files out of this directory at any time. Puppet includes a command to restore the files, but it doesn't provide a command to find them. Given the odd directory structure with the file contents stored by their MD5 checksum, this isn't trivial. Here's a good script for finding the Puppet backups of a given file:

```
#!/bin/bash
FILE=$1
FOUND=0
CLIENTBUCKET=$(puppet config print clientbucketdir)
```

```

echo "Searching for local backups of $FILE..."
for backup in $(find $CLIENTBUCKET -type f -name paths \
  -exec grep -l $FILE {} \; |xargs -r ls -t);
do
  hash=$( basename $(dirname $backup))
  filename=$(< $backup )
  modify_time=$(stat --format '%y' $backup)
  echo -e "$filename\t$hash\t$modify_time"
  FOUND=$((FOUND+1))
done

if [ $FOUND -gt 0 ]; then
  if [ $FOUND -eq 1 ]; then
    echo "1 backup was found."
  elif [ $FOUND -gt 1 ]; then
    echo "$FOUND backups were found."
  fi
  echo ""
  echo "To view a file: puppet filebucket get -b $CLIENTBUCKET
<hash>"
  echo "To restore a file: puppet filebucket restore -b $CLIENTBUCKET
/new/path <hash>"

else
  echo "No previous versions of the $FILE were found."
fi

```

An exercise for the reader: use the file resource example from the *Quick start* section and replicate this script to all client nodes.

## Using environments to test changes

**Environments** provide you with the ability to isolate pieces of code and only invoke them for systems specifically configured for that environment, or only when the environment is enabled by hand. I'll show you how you can use environments to extend what Puppet can accomplish.

The following configuration option will provide flexibility when loading modules. This configuration will first check to see if a module exists in an environment-specific directory, then fall back to the main module directory. The rest of the following examples assume that you have added this option and restarted the Puppet master:

```

[master]
modulepath = $confdir/env/$environment/modules:$confdir/modules

```

## Testing your code

Once you are using puppet in production, you will want to avoid testing new modules on your production systems. Assuming you set the `modulepath` as listed in previous configuration, you can put new the modules you want to test in `/etc/puppet/env/testing/modules`. If you have a dedicated test system, you can put the following in the `puppet.conf`:

```
[agent]
  environment = testing
```

You can also invoke the puppet agent in the testing environment on a case-by-case basis, if you need to test a new module on a particular system:

```
$ sudo puppet agent --test --environment testing
```

Using this approach, you can test out changes to your modules prior to putting the new version in the main module path and affecting hundreds of client systems.

## Release engineering

Another way to use environments is for modules that you do not want to run every time, but only upon request. For example, you may have a module that pushes out a software release.

Assuming you set the `modulepath` as listed before, you can put the new modules you want to test in `/etc/puppet/env/release/modules`. To perform the software release, you would invoke Puppet on that system in the 'release' environment.

```
$ sudo puppet agent --test --environment release
```

## Running the Puppet server under Passenger

The Puppet master process is a WEBrick ruby application server. It will handle only two concurrent Puppet clients. In any environment with more than 50 Puppet clients before it gets overloaded. If you intend to have more than 50 clients, you should disable the Puppet master process and enable it under a more powerful Ruby on the Rails application server. In this section we will document how to make it run under the popular Passenger Rails application server.

First, you will need to install the Apache web server and Phusion Passenger for your platform. The platform-specific instructions are available at <http://www.modrails.com/documentation/Users%20guide%20Apache.html>.

If you are using a Red Hat or CentOS system, it could be as simple as this:

```
$ sudo rpm -i http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
$ sudo rpm -i http://passenger.stealthymonkeys.com/rhel/6/passenger-release.noarch.rpm
$ sudo yum install httpd rubygems mod_ssl mod_passenger
```

Configure the puppetmaster service as a Passenger application. We do this by creating a configuration file for Apache, which defines the Puppet master application, and a `config.ru` file that contains the **Phusion Passenger** application environment. All of these steps must be done as root:

```
$ sudo bash
# sed -e 's/squigley.namespace.at/puppet.example.net/' \
  -e 's#/etc/puppet/ssl#/var/lib/puppet-server/ssl#' \
  /usr/share/puppet/ext/rack/files/apache2.conf \
  > /etc/httpd/conf.d/puppetmaster.conf

# mkdir -p /etc/puppet/rack/public
# sed -e 's#/var/lib/puppet#/var/lib/puppet-server#' \
  /usr/share/puppet/ext/rack/files/config.ru \
  >/etc/puppet/rack/config.ru
# chown puppet /etc/puppet/rack/config.ru
# exit
```

The `chown` command shown in the previous code snippet is essential. Passenger runs the application specified in the `config.ru` file as the user who owns the file. Failing to set the owner of this file correctly is more than 50 percent of the issues we see on the mailing list.

Now, stop the Puppet master service (and Puppet Dashboard if you have it) and start Apache Passenger.

```
$ sudo /sbin/service puppetmaster stop
Stopping puppetmaster: [OK]
$ sudo /sbin/chkconfigpuppetmaster off
$ sudo /sbin/chkconfighttpd on
$ sudo /sbin/service httpd start
Starting httpd: [OK]
```

At this point you can test your clients as before. Absolutely nothing should be different from a client/agent perspective.



To run Puppet Dashboard under Passenger, you would create a configuration file named `/etc/httpd/conf.d/dashboard.conf`, similar to the example file `/usr/share/puppet-dashboard/ext/passenger/dashboard-vhost.conf`. However, be aware that the first 12 lines duplicate the settings already present in the other files in this directory. I would start with the following file, and add more options from the example as necessary:

```
<VirtualHost *:80>
  ServerName dashboard.example.net
  DocumentRoot /usr/share/puppet-dashboard/public/
  RailsBaseURI /
  <Directory /usr/share/puppet-dashboard/public/>
    Options None
    Order allow,deny
    allow from all
  </Directory>
  ErrorLog /var/log/httpd/dashboard.example.net_error.log
  LogLevel warn
  CustomLog /var/log/httpd/dashboard.example.net_access.log combined
  ServerSignature On
</VirtualHost>
```

Then perform the following steps to make the change active:

```
$ sudo /sbin/service puppet-dashboard stop
Stopping Puppet Dashboard:          [OK]
$ sudo /sbin/chkconfig puppet-dashboard off
$ sudo rm /etc/httpd/conf.d/welcome.conf
$ sudo /sbin/service httpd restart
Stopping httpd:                      [OK]
Starting httpd:                      [OK]
```

## Using external data in your Puppet policy

Do you already have a source of data about your hosts or users? You really wouldn't want to maintain that same data in your Puppet policy now, would you? In this section we will talk about how to import external data and use it in your Puppet policy.

The best way to source external data is through a component called **Hiera**. Since you can import any kind of data, this topic is worthy of its own book. But we will set up a very small but powerful example that you can easily expand on: providing a list of user accounts.

All examples in this section use the YAML data format. You can install modules to provide other data backends, like JSON or MySQL. Puppet 3 includes only two Hiera backends by default: YAML and Puppet. The Puppet backend allows you to lookup data from Puppet and return it with the YAML data in a single result set.

## Configuring Hieradata location

First, we need to tell Puppet where to find Hieradata for this class. Assuming we would like to store the Hieradata input data in `/etc/puppet/hiera`, you should create the file `/etc/puppet/hiera.yaml` with the following input. Note that YAML is very exact in its syntax, so you need to type in every dash and every space in this text:

```
---
:backends:
  - yaml
:yaml:
  :datadir: '/etc/puppet/hiera'
:hierarchy:
  - %{:hostname}
  - common
```

This file tells Puppet that the Hieradata format will be YAML, and the files will be in the `/etc/puppet/hiera` directory. Then it tells Puppet to first look for a file with the name of the client host (from the `hostname` fact) with `.yaml` appended to the end, then to look for a file named `common.yaml`.

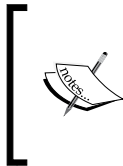
## Creating Hieradata

Next we should create the actual YAML data. Here is an example we could use for `/etc/puppet/hiera/common.yaml`:

```
---
managehomedirs: true
users:
  root:
    uid: 0
    gid: 0
  jack:
    uid: 1002
    gid: 1002
  jill:
    uid: 1001
    gid: 1001
```

If we wanted to add the user `john` to the system named `web1`, we would create another file `/etc/puppet/hiera/web1.yaml`:

```
---
users:
  john:
    uid: 1003
    gid: 1003
```



You might be asking yourself: Isn't this just another file I have to maintain? Yes, in our example, this file is a static text file. In real life, you would have this data be output in the YAML or JSON format from your other application. You can also create a custom Hieradata backend to query the application directly for the data.

## Using Hieradata in a Puppet module

Next we use a function to import data from Hieradata into the Puppet module. Back in our theoretical `users` module, the import functions would look as follows:

```
$managehomedirs = hiera('managehomedirs')
$users          = hiera_hash('users')
```

The first function there will find a single value for `managehomedirs`. With the hierarchy we named in the preceding example, it will look first for a YAML file named for the hostname, then it will look in `common.yaml`. Since this value is defined in the common file, `managehomedirs` will be set to `true`.

The second Hieradata function will return a merged hash of every value for that name in the hierarchy. For any system other than `web1`, the array will contain `root`, `jack`, and `jill`. For the `web1` system it will contain `john`, `root`, `jack`, and `jill`.

Now, here is a complete module for managing the users on your system. You'll note that this module includes no data. All data is sourced from Hieradata, thus allowing Puppet to use another application as a source for policy data:

```
class 'users' {
  $managehomedirs = hiera('managehomedirs', true)
  $defaults       = { managehome => $managehomedirs }
  $userlist       = hiera_hash('users')
  create_resources( user, $userlist, $defaults )
}
```

This module is pretty straightforward:

- ◆ We load in the value for `managehomedirs`. If we cannot find the value, we default to `true`. We then use this as the sole key in a hash of default values.
- ◆ We load in every user defined in every backend in the hierarchy.
- ◆ We invoke the `create_resources()` function to create a user resource for each user defined in our previous Hieradata files.



This is a very powerful module. If you have a thousand user entries in your YAML data, it will create a thousand user resources in the Puppet catalog.

This section has provided you with a quick snapshot into the power and flexibility provided by Hiera data lookups in Puppet. You can create extensive hierarchies for data lookup, and you can use multiple data backends to provide policy data to Puppet. This will allow you to create Puppet policies which are independent of the data and source all of your configuration data from another application you use.

### **Learning more**

You can find more information about using Hiera at <http://puppetlabs.com/blog/first-look-installing-and-using-hiera/>.

You can find more information about the YAML data format at <http://www.yaml.org/>.

## People and places you should get to know

Puppet has a very active developer and user community. This section provides you with many useful links to the project page and mailing lists, as well as a number of helpful articles and tutorials.

### Official site

The official site is the only location to download Puppet itself, and contains the best tutorials for various bits of the puppet ecosystem. The official site is <http://docs.puppetlabs.com/#puppetpuppet>.

### Community support

- ◆ Official mailing list: <http://groups.google.com/group/puppet-users>
- ◆ Official IRC channel: #puppet on the freenode IRC network

### The Puppet Forge

The Puppet Forge is a community-driven collection of custom Puppet modules written by others. Before you set out to write your own module, you should check the Forge to see if someone has already done most of the work for you. The Puppet Forge webpage is at <http://forge.puppetlabs.com/>.

### Articles and tutorials

- ◆ **Learning Puppet:** <http://docs.puppetlabs.com/learning>  
This is a guided tour of learning how to use Puppet. This covers much of the same topics that we've covered in this starter book.
- ◆ **Agent/Master communication:**  
[http://docs.puppetlabs.com/learning/agent\\_master\\_basic.html](http://docs.puppetlabs.com/learning/agent_master_basic.html)  
This is a flowchart of the agent and master communication process, which we have skipped over here. This provides really useful information to help you understand what information is available for use when evaluating a node's catalog.
- ◆ **Puppet and Hiera:**  
<http://puppetlabs.com/blog/first-look-installing-and-using-hiera/>  
You can find more information about using Hiera from this blog entry.

## In-depth details

The following links go into the topics we have introduced in this book in greater depth:

- ◆ **Ordering:** <http://docs.puppetlabs.com/learning/ordering.html>
- ◆ **Conditionals:** <http://docs.puppetlabs.com/learning/variables.html>
- ◆ **Templates:** <http://docs.puppetlabs.com/learning/templates.html>
- ◆ **Class parameters:** <http://docs.puppetlabs.com/learning/modules2.html>
- ◆ **Defined types:** <http://docs.puppetlabs.com/learning/definedtypes.html>

## The Puppet language

- ◆ **Puppet language guide:** [http://docs.puppetlabs.com/puppet/3/reference/lang\\_summary.html](http://docs.puppetlabs.com/puppet/3/reference/lang_summary.html)
- ◆ **Resource types:** <http://docs.puppetlabs.com/references/latest/type.html>
- ◆ **Functions:** <http://docs.puppetlabs.com/references/latest/function.html>
- ◆ **Metaparameters:** <http://docs.puppetlabs.com/references/latest/metaparameter.html>
- ◆ **Report handlers:** <http://docs.puppetlabs.com/references/latest/report.html>

## Related projects

- ◆ **Puppet Dashboard:**  
<http://puppetlabs.com/puppet/related-projects/dashboard/>  
Puppet Dashboard is a product provided by Puppet Labs to store reports in a database and provide a web UI to review those changes.
- ◆ **PuppetDB:** <http://docs.puppetlabs.com/puppetdb/>  
PuppetDB is the fast, scalable, and reliable data warehouse for Puppet. It caches data generated by Puppet, and gives you advanced features with a powerful API.
- ◆ **The Marionette Collective:** <http://puppetlabs.com/mcollective/>  
MCollective is a framework to build server orchestration or parallel job execution systems. It enables real-time discovery of network resources and can select which resources to affect, based on configuration data from leading systems management platforms, including Puppet.





Thank you for buying  
**Instant Puppet 3 Starter**

## About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike.

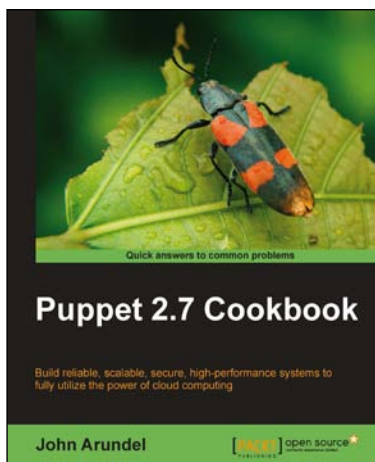
For more information, please visit our website: [www.packtpub.com](http://www.packtpub.com).

## Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to [author@packtpub.com](mailto:author@packtpub.com). If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



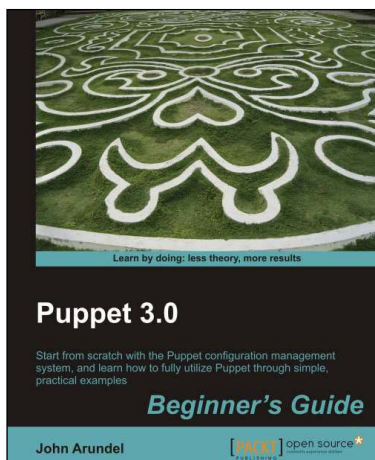


## Puppet 2.7 Cookbook

ISBN: 978-1-84951-538-2      Paperback: 300 pages

Build reliable, scalable, secure, high-performance systems to fully utilize the power of cloud computing

1. Shows you how to use 100 powerful advanced features of Puppet, with detailed step-by-step instructions
2. Covers all the popular tools and frameworks used with Puppet: Dashboard, Foreman, MCollective, and more
3. Includes the latest features and updates in Puppet 2.7



## Puppet 3.0 Beginner's Guide

ISBN: 978-1-78216-124-0      Paperback: 300 pages

Start from scratch with the puppet configuration management system, and learn how to fully utilize Puppet through simple, practical examples

1. Shows you step-by-step how to install Puppet and start managing your systems with simple examples
2. Every aspect of Puppet is explained in detail so that you really understand what you're doing
3. Gets you up and running immediately, from installation to using Puppet for practical tasks in a matter of minutes

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles

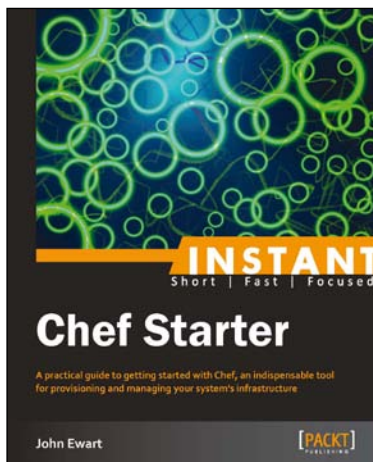


## Microsoft SQL Azure Enterprise Application Development

ISBN: 978-1-84968-080-6 Paperback: 420 pages

Build enterprise-ready applications and projects with SQL Azure

1. Develop large scale enterprise applications using Microsoft SQL Azure
2. Understand how to use the various third party programs such as DB Artisan, RedGate, ToadSoft etc developed for SQL Azure
3. Master the exhaustive Data migration and Data Synchronization aspects of SQL Azure.



## Instant Chef Starter

ISBN: 978-1-78216-346-6 Paperback: 70 pages

A practical guide to getting started with Chef, an indispensable tool for provisioning and managing your system's infrastructure

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Learn the core capabilities of Chef and how it integrates with your infrastructure
3. Set up your own Chef server for managing your infrastructure

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles