Learnig Puppet

Build intelligent software stacks with the Puppet configuration management suite

Jussi Heinonen
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Puppet Development in Isolation'
- A synopsis of the book’s content
- More information on Learning Puppet
Jussi Heinonen is a seasoned systems developer and an open source enthusiast who lives in Cambridge, UK. He has been working for various software businesses in the media and telecommunications sectors since 1998. During this period, he acquired a few Linux certifications, and more recently, in 2014 and 2015, he passed the Puppet Certified Professional exam. He currently works at the Financial Times as a senior integration engineer. His role revolves around building and designing software delivery pipelines that enable developers to create high-quality software quickly and frequently. In his spare time, he likes to spend time with his family and loves to watch games at Arsenal Football Club.
This book is a step-by-step guide to get started with Puppet development, and use Puppet modules as the building blocks to deploy production-ready application clusters in the virtual environment.

The journey begins with the installation of the development environment on the VirtualBox hypervisor and the installation of the Puppet Learning VM that will be used platforms to test and develop Puppet modules.

You will learn how to manage virtual machines and snapshots effectively and enhance the developer's experience with advanced VirtualBox features.

Once the development environment is up and running, this book will focus on Puppet module development in detail. You will be guided through the process of how to utilize the existing modules that are available in the public module repository, write your own modules, and use modules to deploy a real-world web application that includes features such as monitoring and load balancing. When an application cluster is deployed, the focus shifts to how to scale the environment and turn the static configuration into a dynamic one through stored configurations and PuppetDB. The latter part of the book will provide you with practical advice on Puppet troubleshooting, and how to manage your environment with a wealth of features provided by the Puppet Enterprise Console. Starting from the basics, this step-by-step guide will walk you through the process of becoming the master of your own Puppets.
What this book covers

Chapter 1, *Puppet Development in Isolation*, teaches you how to set up the local Puppet development environment quickly and start experimenting with Puppet on the command line in a matter of minutes.

Chapter 2, *Managing Packages in Puppet*, shows you how to restore the virtual machine snapshot and set up shared folders and host-only networking to enhance the developer’s experience. You can also learn how to purge a software package using Puppet.

Chapter 3, *My First Puppet Module*, explains the concept of Puppet modules. You will learn how to install third-party Puppet modules from Puppet Forge and how to create and use your own modules.

Chapter 4, *Monitoring Your Web Server*, teaches you how to use Puppet to install the Nagios monitoring server and how to add a web server to monitor using Puppet.

Chapter 5, *Load Balancing the Cluster*, introduces you to parameterized classes and defined types and teaches you to use these to configure load balancing in the cluster.

Chapter 6, *Scaling Up the Puppet Environment*, introduces you to the Puppet Master, which enables you to centrally manage a large number of Puppet Agent nodes. You will learn how to sign Puppet Agent certificates and join agents to the Puppet environment.

Chapter 7, *Making the Configuration Dynamic*, teaches you how to use Puppet’s Exported Resources to pass Puppet resources between nodes. This chapter also introduces you to the PuppetDB queries that are used to discover services in the cluster.

Chapter 8, *Extending Puppet*, teaches you how to extend Puppet beyond its built-in functionality. This chapter introduces you to custom facts and functions.

Chapter 9, *The Puppet Enterprise Console*, explores the Puppet Enterprise Console, which is a web-based management console that runs on the Puppet Master node. In this chapter, we will learn about Role-based Access Control and how to classify nodes using the External Node Classifier.

Chapter 10, *Troubleshooting Puppet*, teaches you how to identify the most common issues in Puppet and how to tackle them. This chapter provides you with basic troubleshooting skills.
Welcome dear reader. You have arrived at the starting point of the journey to learn Puppet. Whether you have a background in software development, IT infrastructure, or somewhere in between or there about, I believe you have heard people talking about Puppet, and how Puppet can help you automate the configuration management and software deployment processes. I've been using Puppet on a daily basis for the past 4 years, and I feel that it has improved my quality of life at work a lot. I have a background in system administration, and I build software stacks from a set of packages, configuration files, and other types of resources. Prior to Puppet, I used to use various self-written scripts to automate the deployment processes in order to make the process repeatable, but I'm doing much less of that since I discovered Puppet. The problem with scripts, as I see it, is that they are hard to transfer across and to hand over, as scripts are often complex and difficult to read by people who are unfamiliar with the language in which the scripts are written.

Puppet can help you overcome this issue in a two-fold solution:

- Puppet manages resources, such as files, users, and services out of the box. Instead of writing custom Shell scripts to manage resources, we write the Puppet script, which we call the manifest.
- Puppet has its own language called Puppet DSL that is easy to understand by the developers as well as the people involved in the infrastructure.
The moment I start feeling bored with the project I'm working on, because I'm not learning new skills any more, I start to wrap things up, finalize the documentation, and tidy up all the loose ends. The handover process for the project used to involve several days of training followed by a period of several weeks of questions about how the scripts work, and how to change the logic in them. The questions often were as simple as "How do you do this thing in Bash?".

Now the logic has been moved away from the custom scripts to Puppet manifests that are written in Puppet DSL. When I get a question such as "How do I do this in Puppet?", I can reply by saying "Here is a book about Puppet called Learning Puppet. By the end of this chapter, you'll already know how to manage your systems with Puppet". There are dozens of books written on Puppet. This one aims to be a little bit different from those by taking a slightly more practical approach to Puppet development. We will perform the following tasks here:

- Download the Puppet Learning VM
- Take a snapshot of the Learning VM to enable an easy rollback to the original system state
- Start the Learning VM
- Experiment with the Puppet command-line commands: `puppet describe`, `puppet resource`, and `puppet apply`

**Puppet Manifests**

Before we get our hands dirty with Puppet, I'd like to expand the topic a little bit.

As you may already know, Puppet is a configuration management tool that enables you to build application stacks from a set of files that the Puppet community refers to as the manifests.

Manifests are a set of instructions that describes how operating systems and application resources are managed by Puppet and how the system configuration should look like after the manifest has been applied to the system.

Manifests are written in a language called Puppet DSL, where DSL stands for Domain Specific Language. DSL is a commonly used term for programming languages that are not general-purpose languages.

When I write Puppet manifests, I consider it a development process. I call it a development process because the process consists of multiple rounds of iterations during which the manifest evolves.
Here is a simplified overview of the Puppet manifest development process:

For iteration 1, follow the given steps:

Begin by writing the initial manifest that installs a software package > Apply the manifest > Ensure that the package is installed.

For iteration 2, follow the given steps:

Extend the manifest to apply the configuration for the package > Apply the manifest > Ensure that the configuration was correctly applied.

For iteration 3, follow the next steps:

Add the logic to start up the service > Apply the manifest > Ensure that the service started > Finish.

In this example, the development processes had three rounds of iterations, each of them containing a task called Apply the manifest.

Imagine a situation where we write a manifest that creates a Linux user account with a root level access but no password. An account with a root level access is equivalent to a local administrator account on a Windows computer. When you apply the manifest on your computer, Puppet will create a user account on your computer without a password, which makes your computer vulnerable to attacks.

In contrast, if you apply the manifest in an isolated development environment, the configuration change is easy to undo as you can quickly tear down the environment and rebuild it from scratch.

Another reason for developing manifests in isolation is consistency. My choice of operating system is Ubuntu Linux and I run it on MacBook Pro hardware. You may run Mac OS X on the Mac mini, and a friend of mine just downgraded to Windows 7 as she was unhappy with the functionality offered by Windows 8.1.

Each of these operating system flavors will behave slightly differently, although all of them do share similar capabilities such as running a virtualization software.

To ensure that the examples and exercises covered in this book produce consistent results for you and me, we will start our journey by installing the VirtualBox virtualization software package, which enables us to run a set of virtual machines that forms our isolated development environment.
Downloading Oracle VirtualBox

To complete this task, you will need an Internet connection and a web browser such as Mozilla Firefox.

If you prefer to use an alternative virtualization technology, you are free to do so as long as the software supports the following functionalities, and you know how to configure the software to enable the following functionalities:

- The ability to run multiple virtual machines concurrently
- Virtual machine snapshots
- Support for shared folders
- Support for host-only networking
- Support for the Open Virtualization Format (.ovf) and Virtual Machine Disk (.vmdk) file formats

This book is based on Oracle VirtualBox Version 4.3. To ensure that the configuration examples this book provides work well, I'd recommend that you download VirtualBox Version 4.3.

VirtualBox 4.3 can be downloaded for free from the VirtualBox website at https://www.virtualbox.org/wiki/Download_Old_Builds_4_3 (Google: virtualbox download 4.3).

On the download page, you should see a category for the VirtualBox platform packages. Select the download option that is most suitable for your operating system:

- Windows users should download VirtualBox for the Windows hosts
- If your computer runs Apple software, choose VirtualBox for OS X hosts
- If you are a Linux user, you can either download the VirtualBox for Linux hosts or alternatively, you may check the software repositories configured on your system and see whether VirtualBox is made available for your Linux distribution
When you have downloaded the VirtualBox installation package, it is time to install it. Double-click on the installation package that you downloaded, and you will see the installation wizard pop up on the screen.

Install VirtualBox with the default options, and we will take a look at how to configure VirtualBox to optimize it for our development environment.

**Downloading the Puppet Learning VM**

You now should have VirtualBox installed, which can run virtual machines for our isolated development environment.

We will be using the Puppet Learning VM Version 3.7.1, which I've uploaded to an Amazon S3 bucket.

The Puppet Learning VM is a virtual machine image that comes with preinstalled Puppet software. Installing Puppet is not a difficult or too time-consuming task, but it will save us all a little bit of our valuable time if we use the Puppet Learning VM, so we can start experimenting with Puppet quicker.

Follow these steps to complete the download process:

1. To download the Puppet Learning VM Version 3.7.1, go to [http://learning-puppet-packt.s3-website-eu-west-1.amazonaws.com](http://learning-puppet-packt.s3-website-eu-west-1.amazonaws.com), and click on the link that says *To download Puppet Learning VM v.3.7.1 click here*.
2. Once the download is finished, you should find a file called `learn_puppet_centos-6.5-pe-3.7.1-ptb-ovf.zip` in the download folder of your web browser.
3. Double-click on the `learn_puppet_centos-6.5-pe-3.7.1-ptb-ovf.zip` file to open the Zip file archive manager and extract the files to the filesystem.
4. At this stage, I'd recommend that you create a dedicated directory under your home directory that acts as a repository for virtual machine files as well as the Puppet manifest files that we will create later in this book.
5. Once the zip archive has been extracted to the directory, you should find the `learn_puppet_centos-6.5.ovf` and `learn_puppet_centos-6.5-disk1.vmdk` files on the disk.

Next, we will import the virtual machine to the VirtualBox, take a snapshot of it, and then we should be ready to start experimenting with Puppet.
Importing the Puppet Learning VM into VirtualBox

The extracted virtual machine image file has to be imported to VirtualBox before we can launch it. Here are the steps to import the image to VirtualBox:

1. Start the Oracle VM VirtualBox Manager and select Import Appliance from the File menu. This will start the Import Virtual Appliance wizard:

2. Click on the browser button that says Choose a virtual appliance file to import when you hover the mouse pointer over the button. Now you can navigate to the directory where the VirtualBox files were extracted to. On my computer, I extracted the files to the /home/jussi/learning/vm directory, so I'll go to this location and select the file called learn_puppet_centos-6.5.ovf. OVF is a virtual machine template file that is an open standard XML file:
3. Once the file is selected, click on **Open**, then click on **Next**, and you should now be in the **Appliance settings** view:
Here, we can configure the virtual machine settings, such as increasing the amount of memory or adding more processor cores.

We don't need to change the default settings, so let's just click on the **Import** button to start the import process:

![Importing appliance](image)

**Virtual machine snapshots**

While the virtual machine appliance is importing, I'll give you a quick introduction to virtual machine snapshots and how we can use them.

A virtual machine snapshot stores the state of the virtual machine at a particular point of time. We can have multiple snapshots of a virtual machine, and we can easily switch between them.

The example I've just given may give you impression that snapshots are backups. They are not!

Snapshots only contain the data that has changed since the previous snapshot, and therefore, an individual snapshot cannot be used to reconstruct the whole virtual machine. To reconstruct the virtual machine from the snapshot, VirtualBox will need the virtual disk file (*learn_puppet_centos-6.5-disk1.vmdk*), all prior snapshots plus the snapshot you want to restore the state to.

When you create a snapshot of the virtual machine, which we will do shortly, you will tell VirtualBox to start writing changes to a snapshot file instead of the virtual machine disk file.

Every time you create a snapshot, VirtualBox creates a new snapshot file and starts writing changes to it.
Chapter 1

The snapshots are laid out in the following type of tree structure:

![Snapshot tree structure image]

If a virtual machine has only one snapshot and we delete it, then VirtualBox writes changes in the snapshot file onto the disk. If a virtual machine contains more than one snapshot and you delete one, then VirtualBox merges two consecutive snapshots.

Having many snapshots may have an impact on the disk's performance, because the disk operations have to traverse through many snapshots to find the file to make the changes. For disk performance reasons it is recommended that you delete older snapshots when they are no longer needed.

**Snapshot of the virtual machine**

Before we start the virtual machine, we should take a snapshot of it so that we can quickly revert to the point of time where we started.
To take a snapshot of the virtual machine and name it, follow the given steps:

1. First, select the virtual machine, then click on the **Snapshots** button at the top right-hand corner of the Oracle VM VirtualBox Manager window:

   ![Oracle VM VirtualBox Manager with Snapshots button highlighted]

   

   2. In the **Snapshots** view, click on the **Take snapshot** button:

   ![Oracle VM VirtualBox Manager with Take snapshot button highlighted]

   

   Take a snapshot of the current virtual machine state
3. Provide a name for the snapshot, I have named it Base image. Then, click on OK.

![Oracle VM VirtualBox Manager](image)

Now, we have a snapshot that enables us to go back to the previous state of the VM as we make changes to the system configuration.

**Puppet on command line**

Now it is the right time to start the real Puppet work:

1. Select the virtual machine from the list, and click on the Start button at the top of the window.
2. Once the virtual machine has booted up, you should see a login prompt:

```
CentOS release 6.5 (Final)
Kernel 2.6.32-431.el6.i686 on an i686
learning login: ___________
```

3. At the login prompt, type in the user name `root` and hit `Enter`. Then, type in password `puppet` and hit `Enter` again.

4. You have now entered the development environment, where we can start familiarizing with the Puppet commands and manage our system. You are free to play around and change the configuration as much as you like.

5. If you happen to break the environment, you can easily restore the original configuration from the snapshot as we did in the earlier paragraph.

**Puppet version**

The Puppet executable can be run from the command line. Let's begin with confirming which version of Puppet we are running. We can check the version by running the following command:

```
puppet --version
```

At the time of writing, the command run on the Learning VM, it shows the Version as 3.7.3 (Puppet Enterprise 3.7.1).
The first Version number 3.7.3 is for the open source version of Puppet that we are using. The second Version number 3.7.1 is for the Puppet Enterprise version number.

What is the difference between these two versions? The difference between them is how these products are packaged, distributed, and supported.

**The open source Puppet**

The open source Puppet is the community-driven version of Puppet that is developed by the open source community and maintained by Puppet Labs. It can be used and distributed freely.

**The Puppet Enterprise edition**

The Puppet Enterprise edition is a distribution that is developed, maintained, and supported by Puppet Labs, which is the commercial arm behind Puppet.

Companies can purchase the Puppet Enterprise license from Puppet Labs, and in return, Puppet Labs provides support services and software updates for the Puppet Enterprise software bundle that enables companies to get up and running with Puppet quickly.

The Puppet Enterprise edition is free to use in environments that consist of 10 or less Puppet managed hosts.

The environment that we will be building throughout the course of this book will consist of four hosts only, which makes Puppet Enterprise a perfect fit for our goal of learning Puppet.

**Puppet resources**

So now we know how to extract the version using the Puppet command-line utility. Let's shift our focus to the resources next.

Resources in Puppet are known as types. Types are operating system resources such as a file, a user, or a package. There are tens of built-in types in Puppet, and in addition to these, you can create your own custom types to manage resources.

We will learn more about custom types later in Chapter 5, *Load Balancing the Cluster*, but for now we will take a look at the built-in types and see how to use them.

A complete list of available built-in types is available on the Puppet Labs website at http://docs.puppetlabs.com/references/latest/type.html.
Run the `puppet describe --list` command in the Learning VM to list all the built-in types known to Puppet.

The output will contain about 60 resources and their descriptions. To paginate the output, you can extend the command by adding `| less` to the end of it.

Here is the command to view the output page by page:

```
puppet describe --list | less
```

You should now be able to scroll the output up and down using the arrow keys, and you can exit the view by pressing `Q` on the keyboard.

All the Puppet types have attributes that are used to describe the characteristics of the resources we want Puppet to manage.

For example, the type `user` has attributes such as a name for the user name and a password for the user account password.

To list the available attributes for a specific type of resource, you can use the `puppet describe <type>` command. For example, to list the available attributes of a type `user`, you can run the `puppet describe user`, or `puppet describe user | less` command to paginate the output.

If you scroll down the list of attributes, you can find a password attribute that is used to set a password for the user account. Another attribute that you can find on the list is called `ensure`, which defines the state of the user account. The attribute can have three values:

- `present`: This ensures that an account is created unless it already exists
- `absent`: This ensures that the account is removed if it exists
- `role`: This is a specific user attribute of the Unix operating system, such as Oracle Solaris, and therefore it has no meaning when running Puppet on Linux like we are doing

**Managing resources from the command line**

We can manage Puppet resources from the command line using the following syntax: `puppet resource <type> <name> <attribute1>=<value> <attribute2>=<value>`.

Let's create a user called `Elisa` on the system using Puppet. In the Learning VM terminal, type in the following command and and hit the `Enter` key:

```
puppet resource user Elisa ensure=present
```
When the command is executed successfully, it produces the following output:

![Image of command output]

The first line of the output displays a notice confirming that the user account Elisa was created by Puppet. Lines 2-4 show the syntax that we will be using when we declare resources in the Puppet manifest files. The manifest files will be explained more in detail later.

Let's take a look at the syntax line by line:

- **Line 1:** `Notice: /User[Elisa]/ensure: created`
  This displays a confirmation of an action that Puppet has created a user called Elisa.

- **Line 2:** `user { 'Elisa':

  This declares a resource for a type user, which follows the opening curly brace ({), that indicates that the user resource name and optional attributes are to follow. The name 'Elisa': at the end of the line sets the Puppet resource name, which will become the name of the user account. The Puppet resource name must contain a colon at the end.

- **Line 3:** `ensure => 'present'

  This attribute means that a user account must be created unless it already exists.

- **Line 4:** }

  The closing curly brace indicates the end of the resource statement.

The name Elisa on line 2 has two use cases. Firstly, it declares the name of the Puppet resource. Each Puppet resource must have a unique name, otherwise Puppet reports an error.

Secondly, the name Elisa is used as the name of the user account that was created. If the user statement contains a name attribute (alongside the ensure attribute), then the value of the name attribute would become the name of the user account and the name Elisa would only be used as the Puppet resource name.
As the name attribute is omitted, Puppet will use the name for the Puppet resource name as well as for the user account name.

Declaring the following statement in the Puppet manifest would result in the Jakob user account being created in the system, as the name attribute would take priority over the Puppet resource name Elisa:

```puppet
user { 'Elisa':
    ensure => 'present',
    name   => 'Jakob',
}
```

This produces an output that is similar to the following, although the numeric information in the output may vary between the systems:

```
uid=501(Elisa) gid=501(Elisa) groups=501(Elisa)
```

Next, we can remove the account Elisa from the system by setting the ensure attribute value to absent.

```
puppet resource user Elisa ensure=absent
```

Assuming that the command executes successfully, you should see the following output:

```
root@learning:~$ puppet resource user Elisa ensure=absent
```

This output is very similar to how we created the user account Elisa except that line 1 confirms that the user account Elisa was removed and line 3 has the ensure attribute value set to absent, which results in the account being removed when declaring a resource in the Puppet manifest file.

To confirm whether Puppet really removed the account, you can run the command `id Elisa` again, which will confirm that the account no longer exists in the system.
The command `id Elisa` should produce the following output:

```
id: Elisa: No such user
```

Congratulations! You just did two system configuration changes using Puppet. It wasn’t hard, right?

**Puppet dry run**

Sometimes, you want to simulate configuration changes without applying the changes to the system. This can be done by adding the `--noop` parameter after the resource keyword in the Puppet command.

To simulate account creation without creating the account, we can extend the previous user account creation command with the `--noop` option:

```
puppet resource user Elisa ensure=present --noop
```

Line 1 in the output tells us that the user account does not exist in the system, and if you run the following command without the `--noop` option, Puppet would create an account:

```
Notice: /User[Elisa]/ensure: current_value absent, should be present (noop)
user { 'Elisa':
    ensure => 'absent',
}
```

The `--noop` option comes in handy when testing Puppet commands for a syntax. To demonstrate this, we declare an invalid value `removed` for the attribute `ensure`:

```
puppet resource user Elisa ensure=removed --noop
```

Puppet will return an error that tells you that we have used an invalid value for the `ensure` attribute:
Use Puppet to examine the current state of resources

Puppet can also be used to query resources from the system. Information produced by the query can be helpful when we are uncertain about what syntax we should be using to create a resource.

Earlier, we created a user account called Elisa. This account was created without a password, which means that the account cannot be used for interactive logins. Let's recreate the user account and use the password attribute to set a password for the account.

As passwords on Linux are encrypted, we must provide them to the password attribute in the encrypted format.

We know that the user account root on the Learning VM uses the password puppet, but we yet don't know how the password would look like in the encrypted format.

No problem, as we can query the password with the command and then use the encrypted password when recreating the user account Elisa.

The following command shows all the attributes for the root user account:

```
Puppet resource user root
```

The preceding command produces the following output:

```
user { 'root':
  ensure => 'present',
  comment => 'root',
  gid    => '0',
  home   => 'root',
  password => 'S$jr5tnJu$hJ9mC2LmW1xvDjju1M/',
  password_max_age => '999999',
  password_min_age => '0',
  shell  => '/bin/bash',
  uid    => '0',
}
```

Now we can create the user account **Elisa** with the **password** attribute, which is the same as the value of the **password** attribute for the user **root**.

The following command will create the user account **Elisa** that uses the password **puppet**:

```
puppet resource user Elisa ensure=present \  
managehome=true \  
password='$$1$$jrm5tnjw$h8JJ9mCZLmJvIxzDLjw1M/'
```

I've split the command into three lines with the backslash character at the end of the first two lines.

The ```$1$jrm5tnjw$h8JJ9mCZLmJvIxzDLjw1M/``` string is a hash of the password **puppet**. Please note that we have to use single quotes around the password hash ```$1$jrm5tnjw$h8JJ9mCZLmJvIxzDLjw1M/``` because the string contains characters that the Linux command line otherwise interprets as a control character.

Now we can test whether we can log on to the system as the user **Elisa**.
Log out from the system using the `logout` command. Then, log in with the username `Elisa` and password `puppet`. You will see the following welcome screen, which confirms that the login of `Elisa` was successful:

```
This CentOS-based VM comes with Puppet Enterprise pre-installed.
To get started consult the Quest_Guide.pdf contained in the .zip file you extracted this VM from.
You can download the latest Quest Guide and the quest tool at:
  - https://downloads.puppetlabs.com/romguide/
  - A PDF version of the Quest Guide is available in the _site/_ directory after the zip file has been extracted.
To view your current IP address, run `factxr ipaddress_eth0`.
To log in to the Puppet Enterprise console, go to: https://<YOUR_IP_ADDRESS_HERE>
  User: admin
  Password: learningpuppet
Bugs can be reported at: https://tickets.puppetlabs.com/browse/LEARNUM
Questions and Comments: learningum@puppetlabs.com
```

**Puppet is run as a user root**

The root account in Linux is equivalent to an administrator account in the Windows operating system. This is the user account that is commonly used for system configuration changes.

The user account `Elisa` that we created does not have the same amount of privileges as the root account.

To change the system configuration in the protected areas of the operating system, we must run Puppet as a root user.

If you are still logged onto the system as user `Elisa`, you can try creating a user account and see what happens.

As the user `Elisa` is not configured to run Puppet, we will do our test using the `Linux adduser` command.
Let's see what response we get if we try to create a user called Jakob using the user Elisa:

```bash
useradd Jakob
```

The output of the preceding command shows you that the user Elisa did not have sufficient permissions to add a new user to the system:

```
/usr/sbin/useradd: Permission denied
```

To avoid possible permission issues on Puppet managed systems, Puppet runs as a user root that provides Puppet full control of the system to add and remove users, install and uninstall software packages, as well as manage system services.

In the end, Puppet is your new system administrator, which manages the system according to the instructions you have provided from the command line or in the form of the Puppet manifest.

**Puppet DSL and manifests**

I've mentioned Puppet manifests earlier, but I haven't yet explained what manifests are. Puppet manifests are text files that declare one or more Puppet resources. Instead of running Puppet resource commands on the command line, you can declare resources in the manifest file and apply the manifest to the system.

Puppet manifests uses the Puppet Domain Specific Language (DSL) and resource statements in the manifest file, which are described in a syntax that looks very similar to a Hash data type in the Ruby language.

We can use our user account Elisa as a simple example of the Puppet manifest syntax.

First, log out from the user account Elisa by running the `logout` command. Then, log on to the system as a root user and remove the user Elisa from the system with the following command:

```bash
puppet resource user Elisa ensure=absent
```

Then, inspect the state of the user account Elisa with the following commands:

```ruby
puppet resource user Elisa
user { 'Elisa':
   ensure => 'absent',
}
```

In the Ruby language, if we try to declare a hash called User that contains a key Elisa with the value of the ensure attribute as absent, we will declare it using the following syntax:

```ruby
User = { 'Elisa' =>
   { 'ensure' => 'absent' } }
```

If you compare the preceding two code blocks, you can see that the Puppet DSL syntax looks similar to the Ruby language syntax, but it is slightly simpler and easier to read than the Ruby equivalent.

The output of the preceding Puppet resource command is spread across three lines only in order to make it easier for us to read. The Puppet parser that reads the manifest file doesn’t care about the line feed characters.

The preceding user resource can be declared on a single line as follows:

```ruby
user { 'Elisa': ensure => 'absent', }
```

We now have the Puppet DSL representation of the user resource Elisa.

### Managing resources with the puppet apply command

We can apply this resource to the --execute (or -e) puppet apply command, which will remove the user Elisa from the system.

As Elisa no longer exists in the system, let’s change the ensure attribute value to 'present' so that Puppet can create the user Elisa:

```bash
puppet apply --execute "user { 'Elisa': ensure => 'present', }"
```
Puppet will display the following output on the screen:

As you might have noticed, the output of the `puppet apply` command looks different from the `puppet resource` command that we used earlier to create user Elisa. Let's examine the output line by line:

- **Line 1:** Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.12 seconds
  The Puppet report shows that the manifest was compiled successfully in 0.12 seconds. The manifest was compiled for the `learning.puppetlabs.vm` Puppet host. The `learning.puppetlabs.vm` Puppet host is a member of the Puppet environment called production.

- **Line 2:** Notice: /Stage[main]/Main/User[Elisa]/ensure: created
  The Puppet report shows that the user Elisa was created successfully on the system.

- **Line 3:** Notice: Finished catalog run in 0.22 seconds
  The Puppet report shows that the Puppet run was completed successfully in 0.22 seconds.

The preceding three lines relate to the following different stages of the Puppet run:

1. Before Puppet can apply the manifest or a set of manifests, it performs an operation where it compiles a catalogue. A catalogue is a collection of Puppet manifests. During the compilation stage, Puppet looks for possible errors in the manifest files and ensures that the manifests were correctly formatted.

2. Once the catalogue is compiled, Puppet moves on to the second stage where it applies the catalogue to the system.

3. The last step of the process is to produce a report of the results of the Puppet run.
Creating Puppet manifests

We covered the Puppet DSL syntax that is used in the Puppet manifests. Let's try to create a manifest and learn how to apply it to the system.

The simplest way to create a manifest is to use the `puppet resource` command to create the resource definition and redirect the output of the command to the manifest file.

The following are the steps to create a manifest:

1. Use the `puppet resource` command to declare a user resource and redirect the command output to a file using a single greater than character `>` followed by the filename:
   ```
puppet resource user Jakob > user.pp
   ```
   This command won't return any message to the screen as you have redirected the command output to a file called `user.pp`.

2. Before we inspect the contents of the `user.pp` file, let's add another user definition to `user.pp` with the following commands. This time, the output redirection is done using the double greater than characters `>>`. The difference between the single and the double greater than characters is how the output file is managed. The `>` character overwrites the file contents if the file already exists, while the `>>` characters append to the file:
   ```
puppet resource user Markus >> user.pp
   ```

3. Let's take a look at the content of the `user.pp` file. To view the content, we can open the file in the text editor. Linux systems usually come with multiple text editors, such as Vi, but we'll use another editor called Nano, which is easier to use than Vi.

4. You can open the `user.pp` file in the Nano text editor by typing the following command:
   ```
nano user.pp
   ```

5. You will see that the `user.pp` file contains two user definitions: the first definition is for the user Jakob and the second definition is for the user Markus. Currently, both the resources have the `ensure` attribute value as `absent`, which corresponds to the current state of the user accounts on the system.
Here is the content of the file in the Nano text editor:

```yaml
user { 'Jakob':
    ensure => 'absent',
}
user { 'Markus':
    ensure => 'absent',
}
```

6. Using the arrow keys on the keyboard, you can move the cursor around the text file. Change both the `ensure` attribute values to `present`.

7. Once the `ensure` attribute values for both the user resources have been changed, the content of the file should be as follows:

```yaml
user { 'Jakob':
    ensure => 'present',
}
user { 'Markus':
    ensure => 'present',
}
```

8. Now press `Ctrl + X` on the keyboard and save the changes by pressing `Y` and then `Enter`.

Well done! You have just created your first manifest file that manages two resources. Now it's time to apply the manifest with the following command:

```
puppet apply user.pp
```
The following is the output generated by the preceding command:

![Puppet Command Output]

You must have probably noticed that this time we ran the puppet apply command without the `--execute` option. The `--execute` option is only used to provide the manifest content from the command line. Now that we have created the manifest file, and if we want to apply it, the `--execute` option can be omitted. Typically, the `--execute` option is used to pass parameters to the Puppet class that is declared in the manifest. We will discuss the Puppet classes more in detail later on in this book.

**Idempotency**

Let's run the command again, and you will notice the difference in the command output compared to the previous Puppet run:

```
puppet apply user.pp
```

This time, the output is shorter. The lines that notify us that the users Jakob and Markus were created are missing in this Puppet run:

```
Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.14 seconds
Notice: Finished catalog run in 0.27 seconds
```

This is due to the idempotent nature of Puppet. As the users Jakob and Markus already exist in the system, Puppet doesn't attempt to recreate these accounts. Idempotency in Puppet means that you can apply the same manifest as many times as you like, and only when the state of the resource in the system is different from the state of the resource declared in the manifest, will Puppet ensure that the required configuration changes are performed according to the manifest.

To demonstrate how Puppet handles idempotency, we will remove the user Markus with the following command, which we are familiar with:

```
puppet resource user Markus ensure=absent
```
Then, apply the manifest again with the `puppet apply user.pp` command, and you can see that the user Jakob, which we did not remove earlier, does not appear in the output but the user Markus is recreated.

Here is the command again:

```
puppet apply user.pp
```

The output of the command is as follows:

```
Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.15 seconds
Notice: /Stage[main]/Main/User[Markus]/ensure: created
Notice: Finished catalog run in 0.35 seconds
```

**Puppet command line versus Puppet manifests**

So far, we have practiced how to manage system resources from the command line with `puppet resource` command, and also learned how to manage resources with the Puppet manifest and `puppet apply` command. When we start expanding our environment with new hosts and increase the number of resources that Puppet manages on these hosts, you will notice that the Puppet command line doesn't scale very well. The Puppet command line typically manages a single resource, such as user Elisa or user Jakob. Each of these resources was created with its own command. If I have 100 user accounts to be managed, then that would result in the same amount of commands to be run, which would be a very tiring job for anyone to do.

Puppet is a configuration management and automation tool that helps you eliminate repetitive tasks, such as creating 100 user accounts. Instead of running the puppet resource command 100 times, it is better if we add all our users once to a single manifest file, call the file with a puppet apply command, and let the Puppet do the hard work for us. Puppet manifests are types of recipes for your system configuration. Once you have described your system configuration in the form of a manifest, you can easily transfer the recipe onto another host and apply the configuration with a single command.
Managing files and directories with a file resource

The phrase "everything is a file" that is often associated with the Linux operating system makes it an ideal environment for Puppet to manage. Puppet is very good at managing files. Puppet's file resource can create files from static source files. You can define the file content with the `content` attribute, or you can create files with a dynamic content using templates. A file resource can also be used to manage directories and links.

The syntax of a file resource is very similar to the user resource syntax, only the set of available attributes is different. Here is a simple example of how to create an empty directory called `/root/Documents`:

```puppet
file { '/root/Documents':
   ensure => directory;
}
```

The first line defines the type of resource we want Puppet to manage, followed by the name of the directory that Puppet creates.

The `ensure` attribute on line two says that the file resource must be a directory. If we omit the `ensure` attribute, Puppet will create a file instead of a directory.

The closing curly brace } on the third and the last line ends the file resource statement.

Let's do a practical experiment with the file resource and write a manifest file that sets a log in greeting message when the user Jakob logs in. In order to do this, our manifest must fulfill the following two criteria:

- The user Jakob must have a home directory to store the login greeting message
- The user Jakob must have a custom `.bash_profile` file present under the home directory

To start with, let's remove the user Jakob from the system so that we can easily recreate the account with a password, and tell Puppet to create a home directory for the user:

```bash
puppet resource user Jakob ensure=absent
```
Now when the user Jakob has been removed, let's generate a user resource for Jakob and redirect the output to the file called jakobs-login.pp. Again, we use the single > character to create a new file:

```
puppet resource user Jakob > jakobs-login.pp
```

Then, using the >> notation to redirect the output to the jakobs-login.pp file, we can generate the file resource snippet for the .bash_profile file that will be placed under the home directory of Jakob with the following command:

```
puppet resource file /home/Jakob/.bash_profile >> jakobs-login.pp
```

Now that we have the manifest body ready for editing, we can open the jakobs-login.pp file in the Nano editor:

```
nano jakobs-login.pp
```

On opening the file, you should see the following file content:

```plaintext
user { 'Jakob':
    ensure => 'absent',
}
file { '/home/Jakob/.bash_profile':
    ensure => 'absent',
}
```

Let's begin by changing the ensure attribute value from absent to present for the user resource Jakob.

Then, to tell Puppet to create the home directory for the user, we need to use the managehome attribute and set its value to true. The managehome attribute is specific to a user resource, and we can use it to tell Puppet to create a home directory for the user under the /home directory. The home directory is needed to store the .bash_profile file, which we will take a look at shortly.

Finally, to enable Jakob to log in using the password puppet, we should set the encrypted password attribute value to $1$jrm5tnjw$h8JJ9mCZLmJvIxvDLjw1M/.

This is how the user resource for Jakob should look like after the changes:

```plaintext
user { 'Jakob':
    ensure => 'present',
    managehome => true,
    password => '$1$jrm5tnjw$h8JJ9mCZLmJvIxvDLjw1M/',
}
```
Before we move on to the file resource, let's save the changes with Ctrl + X and hit Enter. Then, apply the manifest to the `puppet apply` command:

```
puppet apply jakobs-login.pp
```

If the Puppet run was successful, you should see the following output:

```
Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.63 seconds
Notice: /Stage[main]/Main/User[Jakob]/ensure: created
Notice: /Stage[main]/Main/File[/home/Jakob/.bash_profile]/ensure: removed
Notice: Finished catalog run in 0.35 seconds
```

If we take a look at the third line of the output, we can see that Puppet removed the `/home/Jakob/.bash_profile` file although we had not yet created it. This is because of the `managehome` attribute that we declared for the user Jakob, which results in the Linux environment to create the file when the user is created. Because we haven't yet modified the file resource for `/home/Jakob/.bash_profile` in the manifest, the `ensure` attribute value is `absent`. This results in Puppet removing the file.

Don't worry, as we will now tell Puppet to recreate the file with the content that we specify:

1. Open the `jakobs-login.pp` manifest file using the Nano editor using the following command:
   ```
nano jakobs-login.pp
   ```
2. Using the arrow keys, move to the file resource that currently has the following content:
   ```
   file { '/home/Jakob/.bash_profile':
       ensure => 'absent',
   }
   ```
3. Instead of updating the `ensure` attribute value from `absent` to `present`, we can remove the attribute altogether and replace it with the `content` attribute. To greet the user Jakob with his name when he logs in, we can specify the `content` attribute in the following way:
   ```
   file { '/home/Jakob/.bash_profile':
       content => 'echo Hello $(logname)',
   }
   ```
4. When you are done with the changes, you can save the file using Ctrl + X and press Enter.

5. Now let's apply the most recent changes from the manifest;

   puppet apply jakobs-login.pp

The output is as follows:

Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.22 seconds
Notice:/Stage[main]/Main/File[/home/Jakob/.bash_profile]/ensure: defined content as '{md5}7af0d63debeedf19adb8bb239f5ab36'
Notice: Finished catalog run in 0.53 seconds

6. Now, it is the big moment to test whether our configuration changes work as expected. Log out with the exit command and then log in as user Jakob using the password puppet.

If the configuration changes were successful, you should see the bottom of the login banner, showing the message Hello Jakob.

**Puppet configuration**

So far, we have discussed how to configure a system using Puppet. But what about Puppet's own configuration? Can the Puppet configuration be managed by Puppet itself?

The answer is yes, but if you decide to do so, do it with caution. Test your Puppet configuration changes thoroughly in isolation, and test it multiple times before pushing it into a live environment. It only requires a minor error in your configuration, and your Puppet agents become non-functional.

There are two ways to manage the Puppet configuration. The Puppet configuration can be managed from the command line by running the puppet config commands. Or the configuration can be changed by editing the file in /etc/puppetlabs/puppet/puppet.conf, if you are using the Puppet Enterprise edition as we are doing. In the open source Puppet, the configuration file path is /etc/puppet/puppet.conf.

Let's view the contents of the file with the utility called less, which enables us to browse the file with the arrow keys:

```
less /etc/puppetlabs/puppet/puppet.conf
```
The content of the puppet.conf file is similar to the ini configuration files, which are commonly used with Windows applications. The data structure basically is a key value pair separated by the = equivalence sign.

There are also sections in the configuration file that are marked with the section name wrapped inside the block brackets. The sections are as follows:

- The [main] section
- The [master] section
- The [agent] section

The [main] section contains the configuration that is shared by the [master] and [agent] sections.

The [master] section contains the configuration for the Puppet master, which we will discuss in detail later in this book.

The [agent] section contains the configuration for the Puppet agent, which we have already been using when managing resources on the command line.

When you take a look at the second line in the /etc/puppetlabs/puppet/puppet.conf file, you can see a configuration key called certname with the learning.puppetlabs.vm value. Using the arrow keys, when scrolling down to the [agent] section, we find a key called environment with the value production.

Do you recall seeing these values before? You probably do from the output of the puppet apply command that we ran earlier. Here is the output that I'm referring to:

```
Notice: Compiled catalog for learning.puppetlabs.vm in environment production in 0.15 seconds
```

The Compiled catalog for learning.puppetlabs.vm string and the environment production are defined in the Puppet configuration file. When running Puppet in the standalone mode, as we are at this point, the configuration is not that relevant; but later on in this book, when we link the Puppet Agents with the Puppet Master, we will benefit from knowing how to change the Puppet configuration.

To change the Puppet Agent configuration, we can use the Nano text editor and edit the file manually, but as an alternative, we can use the Puppet command-line utility to change the configuration.

As an exercise, we can change the Puppet Agent's identity with the following command:

```
puppet config set certname brandnew
```
While we are at it, let's change the environment as well. As we are developing Puppet, a suitable environment name for it is `development`, which we can set with the following command:

```
puppet config set environment development
```

Puppet expects to find an environment-specific directory in the filesystem, so let's create one with the following command:

```
mkdir /etc/puppetlabs/puppet/environments/development
```

Now run the `puppet apply user.pp` command, and you can see that the configuration changes have become effective:

```
Notice: Compiled catalog for brandnew in environment development in 0.13 seconds
Notice: Finished catalog run in 0.27 seconds
```

Now we can try changing the configuration manually in the Nano editor. Open the configuration file:

```
nano /etc/puppetlabs/puppet/puppet.conf
```

Press Ctrl + W to search for a `certname` key and replace the `brandnew` value with the `learning.puppetlabs.vm string`.

Then, search for an `environment` key name, and Nano will take you a couple of lines down to the end of the `[main]` configuration section. This line got added when we changed the configuration with the `puppet config set` command. Now repeat the search with Alt + W, and you will find another key called `environment` in the `[agent]` configuration block with the original value `production`. Why duplicate keys? Well, by default, the `puppet config set` command manages the configuration under the `[main]` block of the configuration file. The keys specified in this section will take priority over the configuration in the `[master]` and `[agent]` sections.

So, to revert to the environment value `production`, we can just remove the `environment development` from the `[main]` configuration block.

Once the line has been removed, save the `puppet.conf` file by pressing Ctrl + X, confirm the save operation by pressing Y for Yes, and then press Enter.

To confirm that the configuration changes were successfully applied, we can query specific keys in the configuration file with the `puppet config print` command:

```
puppet config print certname environment
```
The output of the command should show that the configuration was successfully changed. Here is a screenshot of the `puppet config print certname environment` command before and after the change.

---

**Summary**

In this chapter, we thoroughly covered the basics of Puppet, such as how to run Puppet on the command line, and how to use, generate, and edit manifests. We also learned how to get the development environment up and running quickly with a little installation effort using VirtualBox.

In the next chapter, we will be adding a little bit more functionality to VirtualBox, and then start experimenting with the new type of Puppet resources; mainly, let's take a look at how to remove resources with Puppet. This will contribute toward the goal, which is to build a Puppet-managed environment that consists of multiple virtual machines.
Where to buy this book
You can buy Learning Puppet from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.