Comprehensive, hands-on training that solves real world problems

Configuration Management with Puppet

Student Workbook (ROLE)

DO405-RHS6.1-en-1-20151204

CONFIGURATION MANAGEMENT WITH PUPPET

Red Hat Satellite 6.1 DO405
Configuration Management with Puppet
Edition 1 20151204

Authors: George Hacker, Forrest Taylor, Rob Locke, Chen Chang
Editor: Brandon Nolta

Contributors: Susan Lauber, Ricardo Alonso

Document Conventions

Notes and Warnings

Note

"Notes" are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.

Warning

"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.

References

"References" describe where to find external documentation relevant to a subject.

Introduction

Configuration Management with Puppet

The *Configuration Management with Puppet* (DO405) course is part of the Emerging Technology series of courses from Red Hat Training. This course is designed for system administrators and cloud administrators who are intending to implement Puppet as integrated with Red Hat products in an operations environment or a cloud computing environment. This course will cover case studies involving Red Hat products that use Puppet: Red Hat Enterprise Linux OpenStack Platform and Red Hat Satellite. Key Puppet concepts will be introduced, including language constructs, modules, classes, and resources. This course will cover the deployment of Puppet server on Red Hat Enterprise Linux and the deployment of Puppet as a client.

Objectives

- Configure Red Hat Enterprise Linux hosts in a DevOps environment using Puppet.

Audience

- System administrators and cloud operators responsible for the management of systems and cloud client systems on one of Red Hat's products (Red Hat Enterprise Linux OpenStack Platform, Red Hat Satellite).

Prerequisites

- Red Hat Certified Engineer (RHCE) certification or equivalent experience.

Orientation to the Classroom Lab Environment

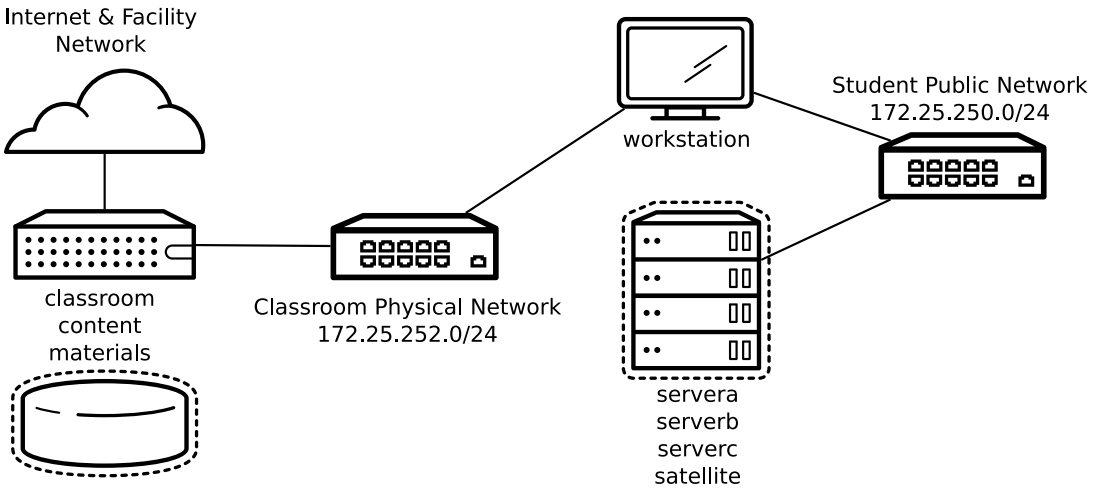

Figure 0.1: Classroom Environment

In this course, students will do most hands-on practice exercises and lab work from a virtual machine called **`workstation`**. The work will have students remotely configuring the other virtual machines called **`servera`**, **`serverb`**, **`serverc`**, and **`satellite`**. All student machines have a standard user account, **`student`**, with the password **`student`**. The **`root`** password on all student systems is **`redhat`**. The **`classroom`** utility server is password-protected from the students.

Classroom Machines

Machine name	IP addresses	Role
`content.example.com` **`materials.example.com`**	172.25.254.254, 172.25.252.254	Classroom utility server
`workstation.lab.example.com` **`workstation.pod`*`X`*`.example.com`**	172.25.250.254, 172.25.252.*X*	Student graphical workstation
`servera.lab.example.com`	172.25.250.10	Student server
`serverb.lab.example.com`	172.25.250.11	Student server
`serverc.lab.example.com`	172.25.250.12	Student server
`satellite.lab.example.com`	172.25.250.8	Student Red Hat Satellite server

The environment has a central utility server, **`classroom.example.com`**, which acts as a NAT router for the classroom network to the outside world. It provides DNS, DHCP, HTTP, and other content services to the students. It uses two alternate names, **`content.example.com`** and **`materials.example.com`**, to provide course content used in the practice and lab exercises. The **`workstation.lab.example.com`** student virtual machine acts as a NAT router between the student network (**`172.25.250.0/24`**) and the classroom physical network (**`172.25.252.0/24`**). **`workstation.lab.example.com`** is also known as **`workstation.pod`*`X`*`.example.com`**, where ***`X`*** in the host name will be a number that will vary from student to student.

Most activities use the **lab** command, executed on **workstation**, to prepare and evaluate the exercise. **lab** takes two arguments: the activity's name and a verb of **setup**, **grade**, **cleanup**, and **solve**.

- The **setup** verb is used at the beginning of an exercise. It will verify that the systems are ready for the activity, possibly making some configuration changes to them.
- The **grade** verb is executed at the end of an exercise. It provides external confirmation that the activity's requested steps were performed correctly.
- The **cleanup** verb can be used to turn back the clock and start the activity over again, usually followed by **setup**.
- The optional **solve** verb can be used to bring the systems to the desired end result state. This can be used for further investigation of the completed activity or to catch up to a later activity that depends on this one.

In a Red Hat Online Learning classroom, students will be assigned remote computers which will be accessed through a web application hosted at *rol.redhat.com* [http://rol.redhat.com]. Students should log into this machine using the user credentials they provided when registering for the class.

Controlling the stations
The top of the console describes the state of the machine.

Machine States

State	Description
none	The machine has not yet been started. When started, the machine will boot into a newly initialized state (the disk will have been reset).
STARTING	The machine is in the process of booting.
STARTED	The machine is running and available (or, when booting, soon will be.)
STOPPING	The machine is in the process of shutting down.
STOPPED	The machine is completely shut down. Upon starting, the machine will boot into the same state as when it was shut down (the disk will have been preserved).
LOCKED	The virtual machine is locked and waiting for the next stage.
TRANSITIONING	The application is moving from one state to another (e.g., from stopped to running).

Depending on the state of the machine, a selection of the following actions will be available to the student.

Machine Actions

Action (button)	Description
Create Application	Create the application. This creates all of the virtual machines needed for the classroom and starts them. This will take several minutes to complete.
Delete Application	Delete the application. This will destroy all virtual machines in the classroom. **Caution: Any work generated on the disk will be lost.**

Action (button)	Description
Start Application	Start all machines in the classroom.
Stop Application	Stop all machines in the classroom.
open console	Open a new tab in the browser and connect to the console of the virtual machine. Log in directly to the machine and run commands. In most cases, log in to the **workstation** machine and use **ssh** to connect to the other virtual machines.
Start	Start ("power on") the machine. Each machine will have a separate **Start** button.
Shutdown	Gracefully shutdown the machine, preserving the contents of its disk. Each machine will have a separate button.
Power Off	Forcefully shutdown the machine, preserving the contents of its disk. This is equivalent to removing the power to a physical machine. Each machine will have a separate button.
Restart	Gracefully shutdown the machine, then start it again (reboot). Each machine will have a separate button.
Redeploy	Forcefully shutdown the machine and reset the disk to its initial state. **Caution: Any work generated on the disk will be lost.** Each machine will have a separate button.
autostop	The timer operates as a "dead man's switch," which decrements as the machine is running. If the timer is winding down to 0, increase the timer if needed.

At the start of a lab exercise, if an instruction to reset **servera** appears, that means the **reset** button in the **servera** console should be pressed. Likewise, if an instruction to reset **workstation** appears, that means press the **reset** button in the **workstation** console.

At the start of a lab exercise, if an instruction to reset all virtual machines appears, that means on the console of each machine, press the **reset** button.

The station timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to two hours when your machine is started.

To adjust the timer, click on the **autostop: ...** link. A **Set Autostop Time** window will open. Set the autostop time in hours and minutes (note: there is a four hour maximum time). Press the **Adjust** button to adjust the time accordingly.

Internationalization

Language support

Red Hat Enterprise Linux 7 officially supports 22 languages: English, Assamese, Bengali, Chinese (Simplified), Chinese (Traditional), French, German, Gujarati, Hindi, Italian, Japanese, Kannada, Korean, Malayalam, Marathi, Odia, Portuguese (Brazilian), Punjabi, Russian, Spanish, Tamil, and Telugu.

Per-user language selection

Users may prefer to use a different language for their desktop environment than the system-wide default. They may also want to set their account to use a different keyboard layout or input method.

Language settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the **Region & Language** application. Run the command **`gnome-control-center region`**, or from the top bar, select **(User)** > **Settings**. In the window that opens, select **Region & Language**. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications, including **`gnome-terminal`**, started inside it. However, they do not apply to that account if accessed through an **`ssh`** login from a remote system or a local text console (such as **`tty2`**).

Note

A user can make their shell environment use the same **`LANG`** setting as their graphical environment, even when they log in through a text console or over **`ssh`**. One way to do this is to place code similar to the following in the user's **`~/.bashrc`** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
  | sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, or other languages with a non-Latin character set may not display properly on local text consoles.

Individual commands can be made to use another language by setting the **`LANG`** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
```

```
jeu. avril 24 17:55:01 CDT 2014
```

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to check the current value of **LANG** and other related environment variables.

Input method settings
GNOME 3 in Red Hat Enterprise Linux 7 automatically uses the **IBus** input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The **Region & Language** application can also be used to enable alternative input methods. In the **Region & Language** application's window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

Once more than one input method is configured, the user can switch between them quickly by typing **Super**+**Space** (sometimes called **Windows**+**Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may find also this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary-shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.

Note

Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl**+**Shift**+**U**, followed by the code point. After **Ctrl**+**Shift**+**U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.

For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl**+**Shift**+**U**, then **03bb**, then **Enter**.

System-wide default language settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, *root* can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it will display the current system-wide locale settings.

To set the system-wide language, run the command **localectl set-locale LANG=*locale***, where *locale* is the appropriate **$LANG** from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in **/etc/locale.conf**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from **Region & Language** and clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the login screen will also adjust the system-wide default language setting stored in the **/etc/locale.conf** configuration file.

Important

Local text consoles such as **tty2** are more limited in the fonts that they can display than **gnome-terminal** and **ssh** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a local text console. For this reason, it may make sense to use English or another language with a Latin character set for the system's text console.

Likewise, local text consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through **localectl** for both local text virtual consoles and the X11 graphical environment. See the **localectl**(1), **kbd**(4), and **vconsole.conf**(5) man pages for more information.

Language packs

When using non-English languages, you may want to install additional "language packs" to provide additional translations, dictionaries, and so forth. To view the list of available langpacks, run **yum langavailable**. To view the list of langpacks currently installed on the system, run **yum langlist**. To add an additional langpack to the system, run **yum langinstall *code***, where *code* is the code in square brackets after the language name in the output of **yum langavailable**.

References

locale(7), **localectl**(1), **kbd**(4), **locale.conf**(5), **vconsole.conf**(5), **unicode**(7), **utf-8**(7), and **yum-langpacks**(8) man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in **localectl** can be found in the file **/usr/share/X11/xkb/rules/base.lst**.

Language Codes Reference

Language Codes

Language	$LANG value
English (US)	en_US.utf8
Assamese	as_IN.utf8
Bengali	bn_IN.utf8
Chinese (Simplified)	zh_CN.utf8
Chinese (Traditional)	zh_TW.utf8
French	fr_FR.utf8
German	de_DE.utf8
Gujarati	gu_IN.utf8
Hindi	hi_IN.utf8
Italian	it_IT.utf8
Japanese	ja_JP.utf8
Kannada	kn_IN.utf8
Korean	ko_KR.utf8
Malayalam	ml_IN.utf8
Marathi	mr_IN.utf8
Odia	or_IN.utf8
Portuguese (Brazilian)	pt_BR.utf8
Punjabi	pa_IN.utf8
Russian	ru_RU.utf8
Spanish	es_ES.utf8
Tamil	ta_IN.utf8
Telugu	te_IN.utf8

CHAPTER 1

IDENTIFYING SYSTEM ADMINISTRATION FUNCTIONS IN PUPPET

Overview	
Goal	Identify system administration functions in Puppet code.
Objectives	• Describe the system administration functions of Puppet code in the OpenStack installers.
Sections	• Identifying System Administration Functions in Puppet (and Quiz)
Lab	• Identifying System Administration Functions in Puppet

Identifying System Administration Functions in Puppet

Objectives

After completing this section, students should be able to:

- Describe the system administration functions of Puppet code in the OpenStack installers.

Finding Puppet in Red Hat Enterprise Linux OpenStack Platform

Puppet allows the definition of what a system should look like using a custom declarative language. These definitions can be created by system administrators to better manage their Unix-like and Microsoft Windows systems. The abstraction offered by Puppet allows administrators to define the configuration in high-level terms without focusing on specific OS-level commands.

Red Hat Enterprise Linux OpenStack Platform is based on OpenStack from the OpenStack Foundation and offers a number of installers:

- **`packstack`**—deploys OpenStack services on pre-installed servers via **`ssh`**.
- **`foreman`**—a Ruby on Rails application that both deploys OpenStack services and adds provisioning of bare metal for enterprise deployments.
- **`director`**—based on the TripleO configuration director, focused on deployment, but adding upgrades and updates for enterprise configuration management.

Demonstration: Installing the Puppet code from RHEL-OSP

The following steps will demonstrate how to install the packages that provide the RHEL-OSP installer. It includes Puppet code that configures the system to support RHEL-OSP.

Resources	
Files	`/etc/yum.repos.d/rhel7osp.repo`
Application URL	`http://materials.example.com/rhel7osp.repo`
Machines	`workstation`

Outcome(s)

You should be able to describe the installation of the RHEL-OSP installers, **`packstack`**, **`foreman`**, and **`director`**, and identify the Puppet code they provide.

1. The Puppet code for each of the installers, **`packstack`**, **`foreman`**, and **`director`**, can be found in their respective **`yum`** repositories. The classroom environment provides a **`yum`** configuration file that provides access to these repositories. It is called **`rhel7osp.repo`** and can be downloaded and installed with the following commands:

```
[root@workstation ~]# cd /etc/yum.repos.d/
[root@workstation yum.repos.d]# wget http://materials.example.com/rhel7osp.repo
--2015-08-06 17:26:40--  http://materials.example.com/rhel7osp.repo
Resolving materials.example.com (materials.example.com)... 172.25.254.254
Connecting to materials.example.com (materials.example.com)|172.25.254.254|:80...
 connected.
HTTP request sent, awaiting response... 200 OK
Length: 149
Saving to: ‘rhel7osp.repo’

100%[======================================>] 149         --.-K/s   in 0s

2015-08-06 17:26:40 (6.97 MB/s) - ‘rhel7osp.repo’ saved [149/149]

[root@workstation yum.repos.d]# cat rhel7osp.repo
[rhel7osp]
name=Red Hat Enterprise Linux OpenStack Platform 7
baseurl=http://content.example.com/puppet3.6/x86_64/dvd/rhel7osp/
enabled=1
gpgcheck=0
[root@workstation yum.repos.d]# cd
[root@workstation ~]#
```

2. List the OpenStack packages that are now available that contain Puppet code.

```
[root@workstation ~]# yum list openstack\*puppet\*
... Output omitted ...
Available Packages
openstack-packstack-puppet.noarch   2015.1-0.11.dev1589.g1d6372f.el7ost rhel7osp
openstack-puppet-modules.noarch     2015.1.8-8.el7ost                   rhel7osp
openstack-tripleo-puppet-elements.noarch
                                    0.0.1-4.el7ost                      rhel7osp
[root@workstation ~]# yum info openstack-packstack-puppet
Loaded plugins: langpacks
Available Packages
Name        : openstack-packstack-puppet
... Output omitted ...
Description : Puppet module used by Packstack to install OpenStack

[root@workstation ~]# yum info openstack-puppet-modules
Loaded plugins: langpacks
Available Packages
Name        : openstack-puppet-modules
... Output omitted ...
Description : A collection of Puppet modules which are required to install and
            : configure OpenStack via installers using Puppet configuration
            : tool.

[root@workstation ~]# yum info openstack-tripleo-puppet-elements
Loaded plugins: langpacks
Available Packages
Name        : openstack-tripleo-puppet-elements
... Output omitted ...
Description : OpenStack TripleO Puppet Elements is a collection of elements for
            : diskimage-builder that can be used to build OpenStack images
            : configured with Puppet for the TripleO program.
```

3. Install all of the OpenStack puppet packages available.

```
[root@workstation ~]# yum -y install openstack\*puppet\*
... Output omitted ...
Transaction Summary
================================================================================
Install  3 Packages (+9 Dependent packages)
... Output omitted ...
Complete!
```

4. Note where the Puppet code has been placed by each of these packages.

```
[root@workstation ~]# rpm -ql openstack-packstack-puppet
/usr/share/openstack-puppet/modules/packstack
... Output omitted ...
[root@workstation ~]# rpm -ql openstack-puppet-modules
/usr/share/openstack-puppet/Puppetfile
/usr/share/openstack-puppet/modules/apache
... Output omitted ...
/usr/share/openstack-puppet/modules/aviator
... Output omitted ...
/usr/share/openstack-puppet/modules/ceilometer
... Output omitted ...
[root@workstation ~]# rpm -ql openstack-tripleo-puppet-elements
/usr/lib/python2.7/site-packages/tripleo_puppet_elements-0.0.1-py2.7.egg-info
... Output omitted ...
/usr/share/doc/openstack-tripleo-puppet-elements-0.0.1
... Output omitted ...
/usr/share/tripleo-puppet-elements
/usr/share/tripleo-puppet-elements/hiera
... Output omitted ...
/usr/share/tripleo-puppet-elements/puppet-modules
... Output omitted ...
```

The Puppet module that configures NTP, provided by the *openstack-puppet-modules* package, will be looked at in the next demonstration.

```
[root@workstation ~]# rpm -ql openstack-packstack-puppet | grep ntp
/usr/share/openstack-puppet/modules/module-collectd/manifests/plugin/ntpd.pp
/usr/share/openstack-puppet/modules/module-collectd/templates/plugin/ntpd.conf.erb
/usr/share/openstack-puppet/modules/ntp
... Output omitted ...
/usr/share/openstack-puppet/modules/ntp/manifests
/usr/share/openstack-puppet/modules/ntp/manifests/config.pp
/usr/share/openstack-puppet/modules/ntp/manifests/init.pp
/usr/share/openstack-puppet/modules/ntp/manifests/install.pp
/usr/share/openstack-puppet/modules/ntp/manifests/params.pp
/usr/share/openstack-puppet/modules/ntp/manifests/service.pp
... Output omitted ...
```

Demonstration: Viewing the Puppet code from RHEL-OSP

The following samples of Puppet code demonstrate the system administration functions that can be performed by Puppet. They Puppet specifications show how the RHEL OpenStack installer uses Puppet to configure the system when it is installed.

Resources	
Files	**/usr/share/openstack-puppet/modules/ntp/manifests/config.pp**, **/usr/share/openstack-puppet/modules/ntp/manifests/init.pp**, **/usr/share/openstack-puppet/modules/ntp/manifests/install.pp**, **/usr/share/openstack-puppet/modules/ntp/manifests/params.pp**, and **/usr/share/openstack-puppet/modules/ntp/manifests/service.pp**
Machines	**workstation**

Outcome(s)

You should be able to identify basic system administration functions in the Puppet code of the RHEL OpenStack installer.

1. The Puppet manifests that configure the NTP service for Red Hat Enterprise Linux OpenStack Platform are found in the **/usr/share/openstack-puppet/modules/ntp/manifests** directory. Changing to that directory first makes it easier to look through the different Puppet manifests that are found in it.

```
[root@workstation ~]# cd /usr/share/openstack-puppet/modules/ntp/manifests
```

2. In the following screenshot, the Puppet *module* is looking to ensure that the *ntp* package is installed. It uses variables to determine whether Puppet should manage the availability of the package, the package name, and the status. Those variables will be found in a later example.

```
[root@workstation manifests]# cat install.pp
#
class ntp::install inherits ntp {

  if $ntp::package_manage {

    package { $ntp::package_name:
      ensure => $ntp::package_ensure,
    }

  }

}
```

3. In the following screenshot, the Puppet module is evaluating the status of the **ntp** service, whether it is running (**ensure**) and enabled on boot (**enable**). Again, the code uses variables that will be viewed in a later example.

```
[root@workstation manifests]# cat service.pp
#
class ntp::service inherits ntp {

  if ! ($ntp::service_ensure in [ 'running', 'stopped' ]) {
    fail('service_ensure parameter must be running or stopped')
  }

  if $ntp::service_manage == true {
    service { 'ntp':
      ensure     => $ntp::service_ensure,
      enable     => $ntp::service_enable,
      name       => $ntp::service_name,
      hasstatus  => true,
      hasrestart => true,
    }
  }

}
```

4. In the following screenshot, the Puppet module is focused on the presence, content, and security of certain configuration files for **ntp**. If keys have been enabled, ensure that the directory for the keys exists, is owned by **root**, and has a permission of **rwxr-xr-x**. Next confirm the configuration file exists, is owned by **root**, has a permission of **rw-r--r--**, and content matches a supplied template.

```
[root@workstation manifests]# cat config.pp
#
class ntp::config inherits ntp {

  if $ntp::keys_enable {
    $directory = ntp_dirname($ntp::keys_file)
    file { $directory:
      ensure => directory,
      owner  => 0,
      group  => 0,
      mode   => '0755',
    }
  }

  file { $ntp::config:
    ensure  => file,
    owner   => 0,
    group   => 0,
    mode    => '0644',
    content => template($ntp::config_template),
  }

}
```

5. In the following screenshot, the Puppet module is defining values for the variables referenced in the previous examples. Note the use of a **case** statement to branch and define different values (from the defaults at the top of the file) based on different supported operating system "families".

```
[root@workstation manifests]# cat params.pp
class ntp::params {
```

```
  $autoupdate        = false
  $config_template   = 'ntp/ntp.conf.erb'
  $disable_monitor   = false
  $keys_enable       = false
  $keys_controlkey   = ''
  $keys_requestkey   = ''
  $keys_trusted      = []
... Output omitted ...

  # Allow a list of fudge options
  $fudge             = []

  $default_config       = '/etc/ntp.conf'
  $default_keys_file    = '/etc/ntp/keys'
  $default_driftfile    = '/var/lib/ntp/drift'
  $default_package_name = ['ntp']
  $default_service_name = 'ntpd'

  $package_manage = $::osfamily ? {
    'FreeBSD' => false,
    default   => true,
  }

  if str2bool($::is_virtual) {
    $tinker = true
    $panic  = 0
  }
  else {
    $tinker = false
    $panic  = undef
  }

  case $::osfamily {
    'AIX': {
      $config        = $default_config
      $keys_file     = '/etc/ntp.keys'
      $driftfile     = '/etc/ntp.drift'
      $package_name  = [ 'bos.net.tcp.client' ]
... Output omitted ...
    'RedHat': {
      $config          = $default_config
      $keys_file       = $default_keys_file
      $driftfile       = $default_driftfile
      $package_name    = $default_package_name
      $service_name    = $default_service_name
      $restrict        = [
        'default kod nomodify notrap nopeer noquery',
        '-6 default kod nomodify notrap nopeer noquery',
        '127.0.0.1',
        '-6 ::1',
      ]
      $iburst_enable   = false
      $servers         = [
        '0.centos.pool.ntp.org',
        '1.centos.pool.ntp.org',
        '2.centos.pool.ntp.org',
      ]
      $maxpoll         = undef
    }
... Output omitted ...
    'Solaris': {
      $config        = '/etc/inet/ntp.conf'
      $driftfile     = '/var/ntp/ntp.drift'
```

```
      $keys_file     = '/etc/inet/ntp.keys'
      $package_name  = $::operatingsystemrelease ? {
        /^(5\.10|10|10_u\d+)$/ => [ 'SUNWntpr', 'SUNWntpu' ],
        /^(5\.11|11|11\.\d+)$/ => [ 'service/network/ntp' ]
      }
      $restrict      = [
        'default kod nomodify notrap nopeer noquery',
        '-6 default kod nomodify notrap nopeer noquery',
        '127.0.0.1',
        '-6 ::1',
      ]
      $service_name  = 'network/ntp'
      $iburst_enable = false
      $servers       = [
        '0.pool.ntp.org',
        '1.pool.ntp.org',
        '2.pool.ntp.org',
        '3.pool.ntp.org',
      ]
      $maxpoll       = undef
    }
... Output omitted ...
```

6. In the following screenshot, the Puppet module is the Puppet class definition file that will reference the previous examples. It is the **`init.pp`** that the RHEL-OSP installers "call" to perform the installation and configuration of OpenStack. Note at the end the references to **`install`**, then **`config`**, and finally **`service`**.

```
[root@workstation manifests]# cat init.pp
class ntp (
  $autoupdate        = $ntp::params::autoupdate,
  $broadcastclient   = $ntp::params::broadcastclient,
  $config            = $ntp::params::config,
  $config_template   = $ntp::params::config_template,
  $disable_auth      = $ntp::params::disable_auth,
  $disable_monitor   = $ntp::params::disable_monitor,
  $fudge             = $ntp::params::fudge,
  $driftfile         = $ntp::params::driftfile,
  $leapfile          = $ntp::params::leapfile,
  $logfile           = $ntp::params::logfile,
  $iburst_enable     = $ntp::params::iburst_enable,
  $keys_enable       = $ntp::params::keys_enable,
  $keys_file         = $ntp::params::keys_file,
  $keys_controlkey   = $ntp::params::keys_controlkey,
  $keys_requestkey   = $ntp::params::keys_requestkey,
  $keys_trusted      = $ntp::params::keys_trusted,
... Output omitted ...
) inherits ntp::params {

  validate_bool($broadcastclient)
  validate_absolute_path($config)
  validate_string($config_template)
  validate_bool($disable_auth)
  validate_bool($disable_monitor)
  validate_absolute_path($driftfile)
  if $logfile { validate_absolute_path($logfile) }
  if $leapfile { validate_absolute_path($leapfile) }
... Output omitted ...
  }

  # Anchor this as per #8040 - this ensures that classes won't float off and
  # mess everything up.  You can read about this at:
```

```
  # http://docs.puppetlabs.com/puppet/2.7/reference/lang_containment.html#known-
issues
  anchor { 'ntp::begin': } ->
  class { '::ntp::install': } ->
  class { '::ntp::config': } ~>
  class { '::ntp::service': } ->
  anchor { 'ntp::end': }

}
```

References

yum(8) and **rpm**(8) man pages

For more information, see

> Puppet 3.6 Reference Manual
> http://docs.puppetlabs.com/puppet/3.6/reference/

Quiz: Identifying System Administration Functions in Puppet

Match the following items to their counterparts in the table.

Specify configuration ownership, permissions, and content.

Specify **ntp** is installed.

Specify **ntp** is running and enabled at boot.

Puppet code	Purpose

`package { $ntp::package_name:`
`  ensure => $ntp::package_ensure,`
`}`	

`file { $ntp::config:`
`  ensure  => file,`
`  owner   => 0,`
`  group   => 0,`
`  mode    => '0644',`
`  content => template($ntp::config_template),`
`}`	

`service { 'ntp':`
`  ensure     => $ntp::service_ensure,`
`  enable     => $ntp::service_enable,`
`  name       => $ntp::service_name,`
`  hasstatus  => true,`
`  hasrestart => true,`
`}`	

Solution

Match the following items to their counterparts in the table.

Puppet code	Purpose

`package { $ntp::package_name:`
`  ensure => $ntp::package_ensure,`
`}`
	Specify **ntp** is installed.

`file { $ntp::config:`
`  ensure  => file,`
`  owner   => 0,`
`  group   => 0,`
`  mode    => '0644',`
`  content => template($ntp::config_template),`
`}`
	Specify configuration ownership, permissions, and content.

`service { 'ntp':`
`  ensure     => $ntp::service_ensure,`
`  enable     => $ntp::service_enable,`
`  name       => $ntp::service_name,`
`  hasstatus  => true,`
`  hasrestart => true,`
`}`
	Specify **ntp** is running and enabled at boot.

Lab: Identifying System Administration Functions in Puppet

In this lab, you will look at Puppet manifests and identify the components found in them.

Resources	
Files	**rht-chrony-0.1.0.tar.gz**
Application URL	**http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz**
Machines	**servera**

Outcome(s)
You should be able to identify some of the basic Puppet functionality in the Red Hat Enterprise Linux 7 OpenStack Platform installer.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-examples setup
```

2. As **student** on **servera**, download the **rht-chrony-0.1.0.tar.gz** Puppet module and extract the files. The module is available at the following URL: **http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz**.

3. Locate and examine the **init.pp** manifest file in the **rht-chrony** Puppet module.

4. How many types of resources are created with this manifest?

5. What permissions are given to the **$driftfile** file?

6. Who owns the **$keyfile** file?

7. Which resource creates a directory?

8. As **root** on **servera**, install the Openstack Puppet related packages.

Solution

In this lab, you will look at Puppet manifests and identify the components found in them.

Resources	
Files	**rht-chrony-0.1.0.tar.gz**
Application URL	**http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz**
Machines	**servera**

Outcome(s)
You should be able to identify some of the basic Puppet functionality in the Red Hat Enterprise Linux 7 OpenStack Platform installer.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-examples setup
```

2. As **student** on **servera**, download the **rht-chrony-0.1.0.tar.gz** Puppet module and extract the files. The module is available at the following URL: **http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz**.

```
[student@servera ~]$ curl -O http://materials.example.com/modules/rht-
chrony-0.1.0.tar.gz
[student@servera ~]$ tar xvf rht-chrony-0.1.0.tar.gz
```

3. Locate and examine the **init.pp** manifest file in the **rht-chrony** Puppet module.

```
[student@servera ~]$ tail -n 18 rht-chrony-0.1.0/manifests/init.pp

    file { $config:
        ensure  => file,
        owner   => 'root',
        group   => 'root',
        mode    => '0644',
        content => template('chrony/chrony.conf.erb'),
        require => File[$driftfile],
    }

    if $service_manage {
        service { $service_name:
            ensure    => $service_ensure,
            enable    => $service_enable,
            subscribe => File[$config],
        }
    }
}
```

4. How many types of resources are created with this manifest?

   There are three types of resources: **package**, **file**, and **service**.

5. What permissions are given to the **$driftfile** file?

0644

6. Who owns the **$keyfile** file?

   root

7. Which resource creates a directory?

   The **file** resource creates all regular files, directories, and symbolic link files. In this manifest, the section **file { $logdir:** creates a file of type directory.

8. As **root** on **servera**, install the Openstack Puppet related packages.

```
[root@servera ~]# yum list openstack\*puppet\*
... Output omitted ...
Available Packages
openstack-packstack-puppet.noarch   2015.1-0.11.dev1589.g1d6372f.el7ost rhel7osp
openstack-puppet-modules.noarch     2015.1.8-8.el7ost                   rhel7osp
openstack-tripleo-puppet-elements.noarch
                                    0.0.1-4.el7ost                      rhel7osp
[root@servera ~]# yum -y install openstack\*puppet\*
... Output omitted ...
Transaction Summary
================================================================================
Install  3 Packages (+9 Dependent packages)
... Output omitted ...
Complete!
```

Summary

In this chapter, you learned:

- Red Hat Enterprise Linux OpenStack Platform installers use Puppet code to configure OpenStack services.
- To read some basic examples of Puppet code used by RHEL-OSP and discern their purpose.

CHAPTER 2

PUPPET ARCHITECTURE

Overview	
Goal	Describe the Puppet architecture and describe a state model.
Objectives	• Describe the Puppet architecture and how best to organize systems. • Describe how Puppet fits into an enterprise environment. • Describe a stateful environment.
Sections	• Puppet Architecture (and Quiz) • Puppet in an Enterprise Environment (and Quiz) • A Stateful Environment (and Quiz)
Review	• Puppet Architecture

Puppet Architecture

Objectives

After completing this section, students should be able to:

- Describe the Puppet architecture and how best to organize systems.

What is Puppet?

Puppet is open source configuration management software. Puppet allows system administrators to write *infrastructure as code* using a descriptive language to configure machines, instead of using individualized and customized scripts to do so. The Puppet domain-specific language (DSL) is used to describe the state of the machine, and Puppet can enforce this state. That means that if an administrator mistakenly changes something on the machine, Puppet can enforce the state and return the machine to the desired state. Thus, not only can Puppet code be used to configure a system initially, but it can also be used to keep the state of the system in line with the desired configuration.

A traditional system administrator would need to log into every machine to perform system administration tasks, but that does not scale well. Perhaps the system administrator would create an initial configuration script (e.g., a Kickstart file) for a machine, but once the initial script has run, the machine can (and will) begin to diverge from that initial script configuration. In the best-case scenario, the system administrator would need to periodically check the machine to verify the configuration. In the worst-case scenario, the system administrator would need to figure out why the machine is no longer functioning or rebuild the machine and redeploy the configuration.

Puppet can be used to set a desired state and have the machine converge to that state. The system administrator no longer needs a script for initial configuration and a separate script (or worse yet, human interaction) for verification of the state. Puppet manages system configuration and verifies that the system is in the desired state. This can scale much better than the traditional system administrator using individual scripts for each system.

Puppet architecture

Puppet uses a server/client model. The server is called a Puppet *master*. The Puppet master stores *recipes* or *manifests* (code containing resources and desired states) for the clients. The clients are called Puppet *nodes* and run the Puppet *agent* software. These nodes normally run a Puppet daemon (**agent**) that is used to connect to the Puppet master. The nodes will download the recipe assigned to the node from the Puppet master and apply the configuration if needed.

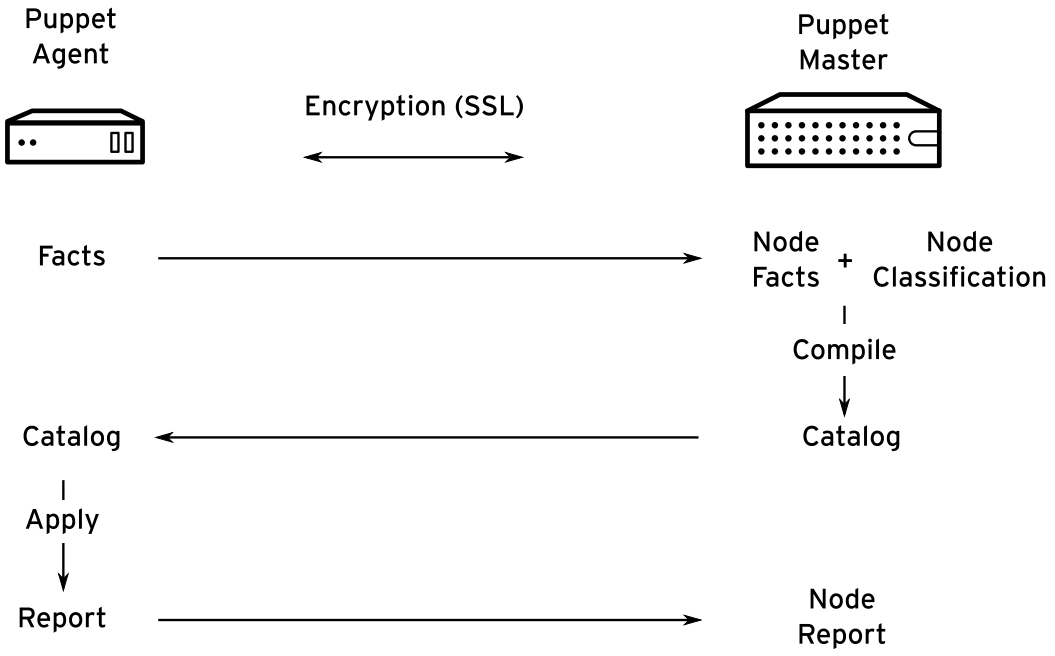

Figure 2.1: A Puppet run

A Puppet run starts with the Puppet node (*not* the Puppet master). By default, the Puppet agent starts a Puppet run every 30 minutes. This run uses secure transmission (SSL) services to pass data back and forth between the Puppet node and the master. The node starts by gathering facts about the system using the **`facter`** command. Facter includes information on block devices, file systems, network interface, MAC addresses, IP addresses, memory, operating system, CPUs, virtualization, etc. These facts are sent from the node to the master.

Once the Puppet master receives the facts from the Puppet node, the Puppet master compiles a *catalog*, which describes the desired state for each resource configured for the node. The Puppet master checks the host name of the node and matches it to the specific node configuration (called *node classification*) or uses the default configuration if the node does not match. This catalog may include dependency information for the resources (e.g., should Puppet install the package first, or start the service first?).

Once the catalog is compiled, the Puppet master sends the catalog to the node. Puppet will then apply the catalog on the Puppet node, configuring all resources defined in the catalog. Puppet is *idempotent*; Puppet can apply the catalog to a node multiple times without affecting the resultant state. Idempotency will be discussed later in the chapter.

Once the catalog is applied to the node by the Puppet agent, the node will report back to the Puppet master with the details of the run. This report includes information on what changes were made to the node (if any), and whether the run completed successfully. Puppet's reporting infrastructure has an API, so other applications can download reports from the Puppet master for storage or further analysis.

What can Puppet manage?

Puppet language is used to describe the desired state of a system using predefined resource types. To a system administrator, these resources will be fairly intuitive. For instance, the

user resource is used to configure users; the **package** resource is used to configure software packages. The resources and Puppet domain-specific language will be discussed in more detail later, but the following snippets of code show some simple DSL code.

```
user { 'notauser':
  ensure => 'absent',
}
```

The previous code will ensure that the user named **notauser** will not be present on the system. If the **notauser** user does not exist on the system, Puppet will report back success. If the **notauser** user exists on the system, Puppet will run the necessary commands to remove the user from the system (**userdel** in this case). Puppet is available on multiple operating systems, so this same code can be used on other supported operating systems without having to know the actual commands to delete a user account. Note that as of Red Hat Satellite 6.1, Satellite does not support running Puppet code on any operating system except Red Hat Enterprise Linux.

```
file { '/var/www/html/index.html':
  ensure => 'file',
  owner  => 'root',
  group  => 'root',
  mode   => '0640',
}
```

The previous code will ensure there is a file named **/var/www/html/index.html** on the system. The owner and group will be **root**, and the mode will be **640**. Although it is not shown in this simple code snippet, Puppet DSL does allow management of file content as well as file ownership and permissions.

```
package { 'httpd':
  ensure => 'installed',
}
```

The previous code will ensure that the **httpd** package is installed on the system.

```
service { 'httpd':
  ensure => 'running',
  enable => true,
}
```

The previous code will ensure that the **httpd** service is started and enabled on the system.

References

For more information on Puppet, see

> Puppet Introduction
> https://docs.puppetlabs.com/puppet/3.6/reference/index.html

For more information on catalog compilation, see

> Puppet Subsystems: Catalog Compilation
> https://docs.puppetlabs.com/puppet/3.6/reference/subsystem_catalog_compilation.html

For more information on the Puppet API, see

> Puppet HTTP API
> https://docs.puppetlabs.com/guides/rest_api.html

Quiz: Puppet Architecture

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. What is the name for the Puppet server?

    a. server
    b. master
    c. marionette
    d. puppeteer

2. What is the name used for the Puppet client software?

    a. client
    b. slave
    c. agent
    d. puppet

3. Which four steps are completed in a Puppet run? (Choose four.)

    a. Gather facts.
    b. Send report to third-party software.
    c. Compile the catalog.
    d. Apply the catalog.
    e. The Puppet master starts the run.
    f. The Puppet agent starts the run.

Solution

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. What is the name for the Puppet server?

   a. server
   b. master
   c. marionette
   d. puppeteer

2. What is the name used for the Puppet client software?

   a. client
   b. slave
   c. agent
   d. puppet

3. Which four steps are completed in a Puppet run? (Choose four.)

   a. Gather facts.
   b. Send report to third-party software.
   c. Compile the catalog.
   d. Apply the catalog.
   e. The Puppet master starts the run.
   f. The Puppet agent starts the run.

Puppet in an Enterprise Environment

Objectives

After completing this section, students should be able to:

- Describe how Puppet fits into an enterprise environment.

Puppet in a DevOps environment

The Puppet agent can be used in many places in an enterprise environment, from the production datacenter to the developer's virtual machine, from the testing environment to the desktop. Puppet is particularly useful in a DevOps environment where developers want to get their code written, tested, and into production many times per day (or per hour). Puppet can be used to manage the configuration of a developer's environment, to install and customize software, to manage accounts, and even to copy in a clean checkout of the code. The *test*, *quality assurance* (QA), or *quality engineering* (QE) environment can be configured exactly like the development environment using Puppet. In fact, using Vagrant in conjunction with Puppet can be an effective method to quickly configure development environments as well as test environments. With a single command, Vagrant will build a virtual machine and Puppet will configure that machine with the desired environment. Once the application or code has passed testing, the code can be deployed into production with a reasonable amount of assurance that the application will successfully launch, especially when the environment in production is configured using the same Puppet configuration as the testing environment.

There are a few things to ensure when using Puppet in a DevOps environment:

- Puppet agents have access to software repositories. If the Puppet configuration specifies the **package** resource to manage packages, the Puppet agent must have access to the proper software repositories (such as the Red Hat Satellite server or the Red Hat subscription repositories).
- Give the environments the same configuration. Puppet can be used to provide exactly the same configuration for machines that are used for development, testing, and production. Sometimes the Puppet master for the development and testing environments are separated from the Puppet master for the production environment (e.g., for security purposes). If that is the case, ensure the development and the testing Puppet configuration match the production Puppet configuration as closely as possible.
- Every Puppet agent should connect to a Puppet master. It is possible to store Puppet configuration locally and have configuration management self-contained, but this method misses out on some benefits of Puppet, namely the reuse of code and central management. Vagrant can be configured to specify a Puppet master at runtime.

In an enterprise environment, it is often desirable to make critical service run in high availability (HA) mode where there is little downtime. Puppet is not able to be configured in HA mode at this time, although there are ways to make a Puppet master redundant or balanced based on load. While the configuration is beyond the scope of this book, the following list itemizes the options available if there are multiple Puppet masters available:

- Statically set the Puppet master. Part of the environment points to one Puppet master while the other part of the environment points to another Puppet master. If one Puppet master goes

down, the Puppet agents can manually change the configuration to point to the other Puppet master.

- Round-robin DNS. DNS is configured to provide different IP addresses for a single host name. This will balance the load between multiple Puppet masters, but the Puppet agents will have trouble if one of the Puppet masters goes down until a DNS update is made.
- Load balancer. A hardware or software load balancer can be used to intelligently redirect requests from the Puppet agents.
- DNS SRV records. This is an experimental feature that provides a pool of Puppet masters in DNS.

Puppet software comes in two varieties: Puppet Enterprise (PE) and the open source Puppet version. Much like Red Hat Enterprise Linux pulls from Fedora as its upstream source, Puppet Enterprise pulls from the open source Puppet version as its upstream source. Puppet Enterprise is supported by PuppetLabs, but the upstream source is not. Although Red Hat Satellite is using the open source version of Puppet, Red Hat has teamed with PuppetLabs to provide support for Puppet within Red Hat Satellite. Red Hat Satellite version 6.1.1 featured integration with Puppet Enterprise (through Puppet Forge).

Quiz: Puppet in an Enterprise Environment

Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which two items should be configured for Puppet in a DevOps environment? (Choose two.)

    a. All Puppet agents need access to software repositories if using the **package** resource.
    b. Multiple Puppet masters for each Puppet agent.
    c. Every Puppet agent points to a Puppet master.
    d. DNS SRV records for the Puppet agents.

2. Which software can be used to provide a virtual environment used for development and testing?

    a. Puppet Enterprise
    b. SRV
    c. CloudPie
    d. Vagrant

Solution

Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which two items should be configured for Puppet in a DevOps environment? (Choose two.)

   a. All Puppet agents need access to software repositories if using the `package` resource.
   b. Multiple Puppet masters for each Puppet agent.
   c. Every Puppet agent points to a Puppet master.
   d. DNS SRV records for the Puppet agents.

2. Which software can be used to provide a virtual environment used for development and testing?

   a. Puppet Enterprise
   b. SRV
   c. CloudPie
   d. Vagrant

A Stateful Environment

Objectives

After completing this section, students should be able to:

- Describe a stateful environment.

A system's state

Puppet's descriptive language is used to describe the state of the system. This state includes but is not limited to:

- Users and groups: Manage users and groups on the system. They can be defined as existing or not existing. All the normal options for users (ID, home directory, shell, etc.) and groups (ID, users, etc.) can be managed by Puppet.
- Files: Ownership, mode, and even content can be managed by Puppet.
- Packages: Manage software packages that are installed (or not installed). Puppet can also manage the version of packages installed.
- Services: Manage services on the system. The state includes running, stopped, enabled, or disabled. Note that the service must be available (installed) before dealing with the service.
- Hosts: Manage entries in the **/etc/hosts** file.
- SSH keys: Manage SSH host keys and SSH authorized keys.

Idempotency

It is possible to run random commands using Puppet, but it is generally discouraged. Puppet needs to be idempotent, which means that regardless of the beginning state, the resultant end state after a Puppet run should always be the same. Consider the following snippet of Puppet code:

```
user { 'notauser':
  ensure => 'absent',
}
```

There are two general states in which the system can be regarding this code before the Puppet run: either the user **notauser** exists on the system or the **notauser** user does not exist on the system. If the user **notauser** exists, the Puppet run will delete the user (using **userdel**). If the user **notauser** does not exist, Puppet does nothing. Either way, after the Puppet run, the user **notauser** will be absent. The Puppet code could be run many times, but the end result will be the same.

Consider the following commands. Which one is idempotent?

```
[root@host ~]# echo '127.0.0.1 localhost' > /etc/hosts
[root@host ~]# echo '127.0.0.1 localhost' >> /etc/hosts
```

The difference is subtle, but important. The first command (using **>**) is idempotent, and will always have an end result of **/etc/hosts** having a single **localhost** line. The second

command (using **>>**) will append to the **/etc/hosts** file, and is not idempotent because the file will be different depending on the number of times the command is run. When using **exec** in Puppet code, ensure that the commands are idempotent.

What Puppet cannot manage

It is important to understand that Puppet uses a stateful language, and thus Puppet only defines the end state of a system. There are many things that Puppet cannot do, which are mostly items that cannot be defined by their state. For example, who made a change to a file? Puppet can ensure that the file has a certain ownership, mode, and even content, but it does not keep track of who changed something on the system. Puppet is not an auditing system; Puppet just ensures the end state of the system. The following list provides some other examples of items that Puppet *cannot* perform.

- Ensure a package was included during installation. This can be provided by Kickstart, not Puppet. However, with Puppet, Kickstart is no longer needed. Every system can start as a minimal installation and use Puppet for the rest of the configuration.
- Why did Bob change this file? What did he change in the file? These are questions that a version control system may be able to answer, not something that is a resultant state that Puppet can configure.

Puppet terms

The following table provides definitions of some common Puppet terms.

Puppet terms

Term	Description
Manifest	A file of Puppet code. Manifest file names usually end in **.pp**, and contain Puppet code.
Module	A structure for Puppet manifests, files, templates, etc. Modules are assigned to Puppet nodes and manage configuration for the node.
Catalog	A compiled set of Puppet code to be applied to a Puppet node. During a Puppet run, the Puppet master compiles a catalog for the Puppet node using the modules assigned to the node. The catalog is applied to the Puppet node to ensure the state.
Resource	An aspect of the system that can be managed by Puppet. This includes files, packages, users and groups, and services. This is the basic unit of Puppet code.
Puppet agent	The Puppet service that runs on the Puppet client. By default, the Puppet agent runs every 30 minutes. The Puppet agent can be run by the system administrator at any time.
Puppet node	The Puppet client.
Puppet master	The Puppet server. The Puppet master can also be a Puppet node to itself or another Puppet master.

References

For more information on Puppet resources, see

> Puppet Language: Resources
> https://docs.puppetlabs.com/puppet/3.6/reference/lang_resources.html

For more information on Puppet architecture, see

> Overview of Puppet's Architecture
> https://docs.puppetlabs.com/puppet/3.6/reference/architecture.html

For more information on idempotency, see

> Introduction to Puppet
> https://docs.puppetlabs.com/guides/introduction.html

Quiz: A Stateful Environment

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which command is idempotent?

   a. **`logger /tmp/file`**
   b. **`echo > /tmp/file`**
   c. **`echo >> /tmp/file`**
   d. **`yum update -y`**

2. Which of the following statements describes an end state?

   a. Ensure a package was included during installation.
   b. Ensure a **`cron`** job runs today.
   c. Ensure Alice logs out at noon.
   d. Ensure the user **`alice`** is present.

Solution

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which command is idempotent?

   a. `logger /tmp/file`
   b. **`echo > /tmp/file`**
   c. `echo >> /tmp/file`
   d. `yum update -y`

2. Which of the following statements describes an end state?

   a. Ensure a package was included during installation.
   b. Ensure a **`cron`** job runs today.
   c. Ensure Alice logs out at noon.
   d. **Ensure the user `alice` is present.**

Quiz: Puppet Architecture

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Where does a Puppet run start?

   a. With the Puppet master.
   b. With the Puppet agent.
   c. With the Puppet proxy server.
   d. With the Puppet catalog.

2. Which three of the following can Puppet manage? (Choose three.)

   a. Files.
   b. Log files.
   c. Packages.
   d. Git repositories.
   e. System services.

3. Which command is idempotent?

   a. **`echo Hello >> /etc/motd`**
   b. **`echo Hello > /etc/motd`**

4. Which statement describes a stateful environment?

   a. George logged into my machine this morning.
   b. The web server configuration file changed.
   c. The *httpd* package is installed.
   d. The **`/etc/httpd/conf/httpd.conf`** file contains my changes from Git.
   e. The web page needs to be updated.

Solution

Choose the correct answer(s) to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Where does a Puppet run start?

   a. With the Puppet master.
   b. With the Puppet agent.
   c. With the Puppet proxy server.
   d. With the Puppet catalog.

2. Which three of the following can Puppet manage? (Choose three.)

   a. Files.
   b. Log files.
   c. Packages.
   d. Git repositories.
   e. System services.

3. Which command is idempotent?

   a. `echo Hello >> /etc/motd`
   b. `echo Hello > /etc/motd`

4. Which statement describes a stateful environment?

   a. George logged into my machine this morning.
   b. The web server configuration file changed.
   c. The *httpd* package is installed.
   d. The `/etc/httpd/conf/httpd.conf` file contains my changes from Git.
   e. The web page needs to be updated.

Summary

In this chapter, you learned:

- Puppet uses a client/server model. The Puppet master is the server, and the Puppet node is the client. The Puppet node runs the Puppet agent.
- A Puppet run starts with the Puppet agent gathering facts on the Puppet node. The facts are sent over SSL to the Puppet master. The Puppet master compiles a catalog and sends the catalog back to the Puppet node. The Puppet node applies the catalog and sends a report to the Puppet master.
- Puppet can manage files, packages, users, groups, and services.
- The Puppet node needs access to software repositories if using the **package** resource.
- A manifest is a file that contains Puppet code. A module contains manifests, files, and templates.
- Idempotency is the ability to set a configuration so the state of the system will be the same, regardless of the starting point.

CHAPTER 3

IMPLEMENTING A PUPPET MANIFEST

Overview	
Goal	Build, validate, and deploy a Puppet manifest.
Objectives	• Describe the Puppet domain-specific language (DSL). • Describe Puppet resources. • Build and apply a simple Puppet manifest. • Implement a Puppet manifest with Geppetto.
Sections	• Puppet Domain-specific Language (DSL) (and Quiz) • Puppet Resources (and Quiz) • Implementing Manifests (and Guided Exercise) • Implementing a Puppet Manifest with Geppetto (and Guided Exercise)
Lab	• Implementing a Puppet Manifest

Puppet Domain-specific Language (DSL)

Objectives

After completing this section, students should be able to:

- Describe the Puppet domain-specific language (DSL).

The Puppet domain-specific language (DSL)

Puppet makes changes to a host to get it to a desired end state. System administrators use the Puppet domain-specific language (DSL) to define what they want their system to be like, not how the changes are made. Puppet provides a layer of abstraction for the desired end state—configure users, files, and services—so that OS-specific implementation details can be ignored. This results in a cross-platform solution.

Machine configurations are specified by manifests. Puppet manifests are text documents written in Puppet DSL that describe the desired end state of the client host.

The Puppet domain-specific language (DSL) was developed to simplify expression of Puppet goals. It was designed to be accessible by system administrators with a limited programming background. Puppet DSL is not a programming language. It is similar to the Ruby programming language, but it is not Ruby.

Puppet DSL has the following syntax:

```
resource_type { 'resource_title':
  attr1 => value1,
  attr2 => value2,
  attr3 => value3,
  ...
  attrN => valueN,
}
```

The resource declaration begins with the type of resource being declared. Puppet includes many built-in resource types such as **`file`**, **`user`**, **`package`**, and **`service`**. The resource type is followed immediately by braces that enclose the resource title and a list of attribute/value pairs. The resource title is a string constant, and string constants are enclosed in single quotes. A colon separates the resource title from the list of attribute assignments.

Each resource type has its own attributes, also called parameters. For example, the **`file`** resource includes the **`ensure`**, **`path`**, **`owner`**, **`group`**, and **`mode`** attributes. These attributes can be assigned values with the **`=>`** operator. Commas separate each of the attribute/value pairs, but administrators typically terminate all attribute/value pairs with commas to make it easier to add additional attribute assignments, if necessary.

Resource title considerations

Care must be taken when defining resource titles. The title is an identifying string that identifies the resource to the Puppet compiler. It does not need to bear any relationship to the actual target host. Resource titles can contain any characters whatsoever, but they are case-sensitive.

Titles must be unique per resource type. There may be a package and a service both titled "http", but there may only be one service titled "http". Duplicate titles will cause a compilation failure.

Multiple similar resources can be abbreviated by specifying a title that is a list of strings. The following example shows how four **file** resources that have the same attributes can be combined into a single resource definition:

```
file { [ '/etc/firewalld',
         '/etc/firewalld/icmptypes',
         '/etc/firewalld/services',
         '/etc/firewalld/zones' ]:
  ensure => 'directory',
  owner  => 'root',
  group  => 'root',
  mode   => '750',
}
```

Puppet DSL shorthand

If an attribute block ends with a semicolon rather than a comma, another title, colon, and attribute block can be specified. Puppet will treat this as multiple resources of a single type. The following example shows how two directory specifications can be combined into a single **file** resource definition:

```
file {
  '/etc/rc.d':
    ensure => 'directory',
    owner  => 'root',
    group  => 'root',
    mode   => '755';

  '/etc/rc.d/init.d':
    ensure => 'directory',
    owner  => 'root',
    group  => 'root',
    mode   => '755';
}
```

References

For more information, see

Puppet 3.6 Reference Manual - Language: Resources
https://docs.puppetlabs.com/puppet/3.6/reference/lang_resources.html

Quiz: Puppet DSL Facts

Choose the correct answer to the following questions:

1. The Puppet DSL is a programming language that administrators use to specify how they want their hosts configured.

   a. True
   b. False

2. Commas are required to separate resource attribute definitions, but they are often used to terminate them as well.

   a. True
   b. False

3. **`'#-of-users'`** is a valid resource title.

   a. True
   b. False

4. An equal sign (=) is used to assign values to attributes in a resource definition.

   a. True
   b. False

5. A resource title can be a list of strings enclosed in brackets, **`[]`**.

   a. True
   b. False

Solution

Choose the correct answer to the following questions:

1. The Puppet DSL is a programming language that administrators use to specify how they want their hosts configured.

   a. True
   b. False

   Although Puppet DSL describes how to configure a machine, it is not a programming language.

2. Commas are required to separate resource attribute definitions, but they are often used to terminate them as well.

   a. True
   b. False

3. **`'#-of-users'`** is a valid resource title.

   a. True
   b. False

   Resource titles are strings that can be composed of any characters.

4. An equal sign (=) is used to assign values to attributes in a resource definition.

   a. True
   b. False

   The **=>** operator is used to assign values in Puppet DSL.

5. A resource title can be a list of strings enclosed in brackets, **[]**.

   a. True
   b. False

   This form allows very similar resources to be abbreviated in a single definition.

Puppet Resources

Objectives

After completing this section, students should be able to:

- Describe Puppet resources.
- Define default resource attribute values.

Puppet resources

Puppet resources include system objects, such as files, users, groups, packages, services, and SSH keys. The Puppet language allows administrators to manage system resources in an operating system independent manner. Puppet's Resource Abstraction Layer (RAL) provides libraries that implement low level functionality on particular operating systems.

The following Puppet resource declaration specifies that the **`httpd`** service should be running and persistently enabled on the system.

```
service { 'httpd':
  ensure => 'running',
  enable => true,
}
```

The responsibility of Puppet's RAL is to bring a particular system to that state. The following commands would implement this on a Red Hat Enterprise Linux 6 host.

```
service httpd start ; chkconfig httpd on
```

On a Red Hat Enterprise Linux 7 system, the following commands would be used instead.

```
systemctl start httpd ; systemctl enable httpd
```

The Resource Abstraction Layer uses the appropriate commands for the system it is running on. This allows Puppet administrators to manage resources at a high level without worrying about the particulars of the underlying operating system. Implementation-specific details are handled by RAL, so the Puppet administrator can write system-independent specifications.

The `puppet resource` command

The **`puppet resource`** command displays Puppet DSL that describes the current state of an object on the local system. Using this command to display the DSL of existing objects displays the attributes (with their current values) of a resource. This will help an administrator learn how to describe their own resources with the Puppet DSL.

The **`puppet resource --type`** command lists the available resource types on a Puppet host.

```
[root@workstation ~]# puppet resource --type
anchor
augeas
computer
cron
```

```
exec
file
file_line
... Output omitted ...
```

The following section will use the **puppet resource** command repeatedly to display the attributes of resources on a Red Hat Enterprise Linux system.

Examples of resources

notify resource

The **notify** resource is used to generate output in the Puppet log. The following example shows how to display the message "Enabling NTP service" in the log:

```
notify { 'log-ntp-enable' :
  message => 'Enabling NTP service',
}
```

package resource

The **package** resource defines whether a package should be installed on the host operating system. The **ensure** attribute should be set to the value **'absent'** when a package should be removed. Its value should be **'present'** when a package should be installed.

```
[root@workstation ~]# rpm -q vsftpd
package vsftpd is not installed
[root@workstation ~]# puppet resource package vsftpd
package { 'vsftpd':
  ensure => 'absent',
}
[root@workstation ~]# rpm -q coreutils
coreutils-8.22-11.el7.x86_64
[root@workstation ~]# puppet resource package coreutils
package { 'coreutils':
  ensure => '8.22-11.el7',
}
```

The **puppet resource package** command displays the current version of an installed package as the value of the **ensure** attribute.

file resource

The **file** resource defines the attributes and content of a file, or directory, on the Puppet host. The **ensure** attribute is set to **'file'** or **'directory'**, or if it has the value **'present'**, the **type** attribute can define its type. The **owner** and **group** attributes define the ownership and group access of the file. The **mode** attribute defines the permissions of the file or directory.

```
[root@workstation ~]# puppet resource file /etc/notafile
file { '/etc/notafile':
  ensure => 'absent',
}
[root@workstation ~]# puppet resource file /root/anaconda-ks.cfg
file { '/root/anaconda-ks.cfg':
  ensure   => 'file',
  content  => '{md5}e08d48c9c39de018140f5f9e2dd99add',
  ctime    => '2015-02-03 17:38:25 -0500',
  group    => '0',
```

```
  mode     => '600',
  mtime    => '2015-02-03 17:38:25 -0500',
  owner    => '0',
  selrange => 's0',
  selrole  => 'object_r',
  seltype  => 'admin_home_t',
  seluser  => 'system_u',
  type     => 'file',
}
[root@workstation ~]# puppet resource file /var
file { '/var':
  ensure   => 'directory',
  ctime    => '2015-08-03 00:10:49 -0400',
  group    => '0',
  mode     => '755',
  mtime    => '2015-08-03 00:10:43 -0400',
  owner    => '0',
  selrange => 's0',
  selrole  => 'object_r',
  seltype  => 'var_t',
  seluser  => 'system_u',
  type     => 'directory',
}
```

`service` resource

The **`service`** resource defines the current and boot-time state of a system service. The **`ensure`** attribute can take the value **`'running'`** or **`'stopped'`** and defines the current state of a service. The **`enable`** attribute defines its boot-time behavior.

```
[root@workstation ~]# puppet resource service sshd
service { 'sshd':
  ensure => 'running',
  enable => true,
}
[root@workstation ~]# puppet resource service nfsd
service { 'nfsd':
  ensure => 'stopped',
  enable => false,
}
```

`user` resource

The **`user`** resource defines which user accounts should be present, or absent, on a host. The attributes of the **`user`** resource specify traits of the system user. The **`uid`** and **`gid`** attributes are set to numeric values that correspond to the user ID and group ID, respectively. The **`home`** attribute defines the user's home directory.

Note

Although the **`home`** attribute specifies the home directory of a user, it does not actually create the directory. That must be specified with a **`file`** resource.

The encrypted password hash for a user is specified with the **`password`** attribute. Additional attributes can be specified that define the user's password aging values.

```
[root@workstation ~]# puppet resource user notauser
user { 'notauser':
```

```
  ensure => 'absent',
}
[root@workstation ~]# puppet resource user root
user { 'root':
  ensure           => 'present',
  comment          => 'root',
  gid              => '0',
  home             => '/root',
  password         => '$6$QLji1eoc$ZJhjFcPu6EJBEyls//8pGgfskDOe0PWrLfcXVkmajAnqZsm4xD0I/
g236TMNjbQAER3FXcfkhlSwSO76qPqx5/',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '0',
}
```

exec resource

The **exec** resource is used to execute a command on the Puppet host. The command to be executed can be specified as the title of the resource or as the **command** attribute. Either the full path name of the executable must be specified, or a **path** attribute can be defined as a list of directories to be searched for the command. The **cwd** attribute specifies the current working directory to be in when the command is executed.

Normally the **exec** resource will execute the command silently. Puppet will display output if the command returns an error. When the **logoutput** attribute is set to **true**, Puppet will display all of the output of the executed command.

OpenStack manifests revisited

Puppet DSL was introduced at the start of this course. Take another look at some of the Puppet resources used by the OpenStack installer.

The following manifest specifies a **package** resource. It uses variables to specify the package name (the resource title) and whether the package should be installed (the **ensure** attribute).

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/manifests/install.pp
#
class ntp::install inherits ntp {

  if $ntp::package_manage {

    package { $ntp::package_name:
      ensure => $ntp::package_ensure,
    }

  }

}
```

The following Puppet manifest uses a **service** resource to define how the NTP network service is configured. It uses variables to specify the service name (the **name** attribute), whether the service should be running, and whether it should be enabled at boot-time.

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/manifests/service.pp
#
class ntp::service inherits ntp {

  if ! ($ntp::service_ensure in [ 'running', 'stopped' ]) {
    fail('service_ensure parameter must be running or stopped')
```

```
  }

  if $ntp::service_manage == true {
    service { 'ntp':
      ensure     => $ntp::service_ensure,
      enable     => $ntp::service_enable,
      name       => $ntp::service_name,
      hasstatus  => true,
      hasrestart => true,
    }
  }

}
```

Default resource values

Puppet has default values for resource attributes. For example, **`file`** resources create ordinary files owned by **`root`** with a default permission of 0644. Directories have a default permission of 0755 when they are created. When attributes are explicitly specified in a resource, they override the default values.

The Puppet DSL has a mechanism for defining different default values for a resource. This is accomplished by specifying the resource type capitalized, followed by an untitled stanza of attributes with new default values. The following Puppet DSL defines new default values for the **`ensure`**, **`owner`**, **`group`**, and **`mode`** attributes of the **`file`** resource.

```
File {
  ensure => 'file',
  owner  => 'apache',
  group  => 'apache',
  mode   => '640',
}
```

References

For more information, see

> Puppet 3.6 Reference Manual - Language: Resources
> https://docs.puppetlabs.com/puppet/3.6/reference/lang_resources.html

For more information, see

> Puppet 3.6 Reference Manual - Language: Resource Defaults
> https://docs.puppetlabs.com/puppet/3.6/reference/lang_defaults.html

Quiz: Puppet Resources

Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. This Puppet resource determines if software is installed on a host.

   a. **`software`**
   b. **`service`**
   c. **`file`**
   d. **`package`**
   e. None of the above

2. This Puppet resource should be used to create a user account on a host.

   a. **`account`**
   b. **`user`**
   c. **`exec`**
   d. **`service`**
   e. None of the above

3. This Puppet resource is used to display messages to the Puppet log.

   a. **`log`**
   b. **`message`**
   c. **`notify`**
   d. **`exec`**
   e. None of the above

4. This Puppet resource executes a specific command on a Puppet host.

   a. **`command`**
   b. **`file`**
   c. **`execute`**
   d. **`run`**
   e. None of the above

5. This resource attribute causes a service to run immediately.

   a. **`ensure`**
   b. **`enable`**
   c. **`execute`**
   d. **`run`**
   e. None of the above

6. This resource attribute affects user access to a file.

a. **`access`**
b. **`perms`**
c. **`permission`**
d. **`mode`**
e. None of the above

7. Omitting a resource title in a resource definition...

a. is permitted because titles are optional.
b. will produce a syntax error.
c. defines default values for attributes for that type of resource.

Solution

Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. This Puppet resource determines if software is installed on a host.

   a. `software`
   b. `service`
   c. `file`
   d. `package`
   e. None of the above

2. This Puppet resource should be used to create a user account on a host.

   a. `account`
   b. `user`
   c. `exec`
   d. `service`
   e. None of the above

3. This Puppet resource is used to display messages to the Puppet log.

   a. `log`
   b. `message`
   c. `notify`
   d. `exec`
   e. None of the above

4. This Puppet resource executes a specific command on a Puppet host.

   a. `command`
   b. `file`
   c. `execute`
   d. `run`
   e. None of the above

5. This resource attribute causes a service to run immediately.

   a. `ensure`
   b. `enable`
   c. `execute`
   d. `run`
   e. None of the above

6. This resource attribute affects user access to a file.

   a. `access`

b. `perms`
c. `permission`
d. `mode`
e. None of the above

7. Omitting a resource title in a resource definition...

a. is permitted because titles are optional.
b. will produce a syntax error.
c. defines default values for attributes for that type of resource.

Implementing Manifests

Objectives

After completing this section, students should be able to:

- Build and apply a simple Puppet manifest.

Installing Puppet software

The Puppet software must be installed on a system before it can be used to apply Puppet DSL manifests. On Red Hat systems, this is accomplished by installing the *puppet* package. A system must be subscribed to one of the following Red Hat repositories to install Puppet software:

- Red Hat Enterprise Linux 7 Server - RH Common (RPMs)
- Red Hat Satellite Tools 6.1 (for RHEL 7 Server) (RPMs)
- Red Hat Ceph Storage Tools 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)

Using Puppet manifests

Puppet manifest files are simple to create. Use a text editor to create a file that ends with a **.pp** extension that defines resource declarations. Apply the changes by running **puppet** with the **apply** subcommand. The following manifest, **nojack.pp**, will cause Puppet to delete an account named **jack** from the system.

```
[root@workstation ~]# id jack
uid=501(jack) gid=501(jack) groups=501(jack)
[root@workstation ~]# vim nojack.pp
[root@workstation ~]# cat nojack.pp
user {'jack':
  ensure => 'absent',
}
[root@workstation ~]# puppet apply nojack.pp
Notice: Compiled catalog for workstation.lab.example.com in environment production
in 0.58 seconds
Notice: /Stage[main]/Main/User[jack]/ensure: removed
Notice: Finished catalog run in 1.02 seconds
[root@workstation ~]# id jack
id: jack: No such user
```

The following manifest, **katello-file.pp**, declares that a file named **/var/tmp/katello-file** should exist. The file ownership, permissions, and content are specified in the resource declaration.

```
[root@workstation ~]# vim katello-file.pp
[root@workstation ~]# cat katello-file.pp
file {'katello-file':
  path    => '/var/tmp/katello-file',
  ensure  => 'file',
  mode    => '640',
  owner   => 'katello',
  group   => 'katello',
  content => "I'm a test file.\n",
}
```

```
[root@workstation ~]# ls /var/tmp
scl0r3DGi  sclyCEuvj
[root@workstation ~]# puppet apply katello-file.pp
Notice: Compiled catalog for workstation.lab.example.com in environment production
in 0.37 seconds
Notice: /Stage[main]/Main/File[katello-file]/ensure: defined content as
'{md5}15735e435ae2af37745ce7f9f0e0fe94'
Notice: Finished catalog run in 0.32 seconds
[root@workstation ~]# ls /var/tmp
katello-file  scl0r3DGi  sclyCEuvj
[root@workstation ~]# ls -l /var/tmp/katello-file
-rw-r-----. 1 katello katello 17 Aug 28 08:52 /var/tmp/katello-file
[root@workstation ~]# cat /var/tmp/katello-file
I'm a test file.
```

Notice how double quotes were used to enclose the string with an apostrophe and a newline character specified as "\n."

References

puppet-apply(8) man page

Guided Exercise: Implementing and Applying Puppet Manifests

In this lab, you will build a simple "Hello world" Puppet manifest and apply it to the **servera** node. You will expand the manifest by adding the following resources: **file**, **package**, **service**, and **user**.

Resources	
Files	**/var/www/html/index.html**
Machines	**servera**

Outcome(s)
You should be able to write a simple Puppet manifest that can install and deploy a network service.

Before you begin
You should have a Red Hat Enterprise Linux server installed with **yum** configured.

1. Log in as **root** on **servera**, using the password **redhat**, and install the *puppet* package.

```
[root@servera ~]# yum -y install puppet
```

2. Build a simple "Hello world" Puppet manifest called **test.pp**. Apply it to the current node, **servera**.

   2.1. Use a text editor to create **test.pp**. Include Puppet DSL that uses a **notify** resource to display "Hello world!". The resulting file should look like the following:

```
# My first Puppet DSL
notify { 'hello-world':
  message => 'Hello world!',
}
```

   2.2. Use the **puppet apply** command to apply the manifest.

```
[root@servera ~]# puppet apply test.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.02 seconds
Notice: Hello world!
Notice: /Stage[main]/Main/Notify[hello-world]/message: defined 'message'
 as 'Hello world!'
Notice: Finished catalog run in 0.04 seconds
[root@servera ~]#
```

3. Expand the manifest by adding resources. Start by creating the **webmaster** user, then apply your changes.

   3.1. Edit **test.pp** and add a **user** resource that creates a user called **webmaster**. The home directory for this user should be **/home/webmaster** and the user's shell should be **/bin/bash**. The resulting file should look like the following:

```
notify { 'hello-world':
  message => 'Hello world!',
}

user { 'webmaster':
  ensure  => 'present',
  home    => '/home/webmaster',
  shell   => '/bin/bash',
}
```

3.2. Confirm that the user and home directory do not previously exist.

```
[root@servera ~]# ls /home/webmaster
ls: cannot access /home/webmaster: No such file or directory
[root@servera ~]# id webmaster
id: webmaster: no such user
```

3.3. Apply the changes introduced in the updated manifest.

```
[root@servera ~]# puppet apply test.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.21 seconds
Notice: Hello world!
Notice: /Stage[main]/Main/Notify[hello-world]/message: defined 'message'
 as 'Hello world!'
Notice: /Stage[main]/Main/User[webmaster]/ensure: created
Notice: Finished catalog run in 0.17 seconds
```

3.4. Check the state of the system to confirm the changes were applied.

```
[root@servera ~]# id webmaster
uid=1001(webmaster) gid=1001(webmaster) groups=1001(webmaster)
[root@servera ~]# ls /home/webmaster
ls: cannot access /home/webmaster: No such file or directory
```

4. Notice the home directory, **/home/webmaster**, was not created. Modify the manifest so Puppet will create it.

4.1. Add a **file** resource to **test.pp** that will create **/home/webmaster**. It should be owned by **webmaster** and have **drwxrwx---** permissions. Since the **webmaster** user should exist, specify a **require** attribute to declare that dependency. The additional DSL should be similar to the following:

```
file { '/home/webmaster':
  ensure   => 'directory',
  owner    => 'webmaster',
  group    => 'webmaster',
  mode     => '770',
  require  => User['webmaster'],
}
```

4.2. Apply the changes and confirm they were performed.

```
[root@servera ~]# puppet apply test.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.36 seconds
Notice: Hello world!
Notice: /Stage[main]/Main/Notify[hello-world]/message: defined 'message'
 as 'Hello world!'
Notice: /Stage[main]/Main/File[/home/webmaster]/ensure: created
Notice: Finished catalog run in 0.05 seconds
[root@servera ~]# ls -ld /home/webmaster
drwxrwx---. 2 webmaster root 6 Jul 31 17:31 /home/webmaster
```

5. Modify the manifest so Puppet will install the web server software, *httpd*, and activate the corresponding service.

   5.1. Add a **package** resource to **test.pp** that will install *httpd*. Add a **service** resource to activate the **httpd** daemon, currently and at boot time. Since the service requires the *httpd* to be installed, specify a **require** attribute to declare that dependency. The additional DSL should be similar to the following:

```
package { 'httpd':
  ensure  => 'present',
}

service { 'httpd':
  ensure  => 'running',
  enable  => true,
  require => Package['httpd'],
}
```

   5.2. Apply the manifest and confirm the changes were made to the node.

```
[root@servera ~]# puppet apply test.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.16 seconds
... Output omitted ...
Notice: Hello world!
Notice: /Stage[main]/Main/Notify[hello-world]/message: defined 'message'
 as 'Hello world!'
Notice: /Stage[main]/Main/Package[httpd]/ensure: created
Notice: /Stage[main]/Main/Service[httpd]/ensure: ensure changed 'stopped'
 to 'running'
Notice: Finished catalog run in 5.73 seconds
[root@servera ~]# rpm -q httpd
httpd-2.4.6-31.el7.x86_64
[root@servera ~]# systemctl is-active httpd
active
[root@servera ~]# systemctl is-enabled httpd
enabled
```

Implementing a Puppet Manifest with Geppetto

Objectives

After completing this section, students should be able to:

- Deploy Geppetto on a Red Hat Enterprise Linux workstation.
- Implement a Puppet manifest with Geppetto.

Puppet Labs publishes a Puppet DSL editor called Geppetto. Geppetto is an integrated development environment (IDE) that can be used to create Puppet manifests and modules. Geppetto can be used by a single administrator, or it can be used to collaborate with others. It can be integrated with revision control software, such as **`git`** or Subversion. Geppetto can also publish Puppet modules directly to Puppet Forge if authentication credentials are provided.

Installing Geppetto

There are currently two methods for installing Geppetto. One method downloads and installs Geppetto as a standalone product. The other method starts with an existing Eclipse IDE installation and adds Geppetto functionality.

Standalone Geppetto Installation

The Puppet Labs Geppetto home page has a section that describes how to install Geppetto. It contains a hyperlink that points to the Geppetto software download page.

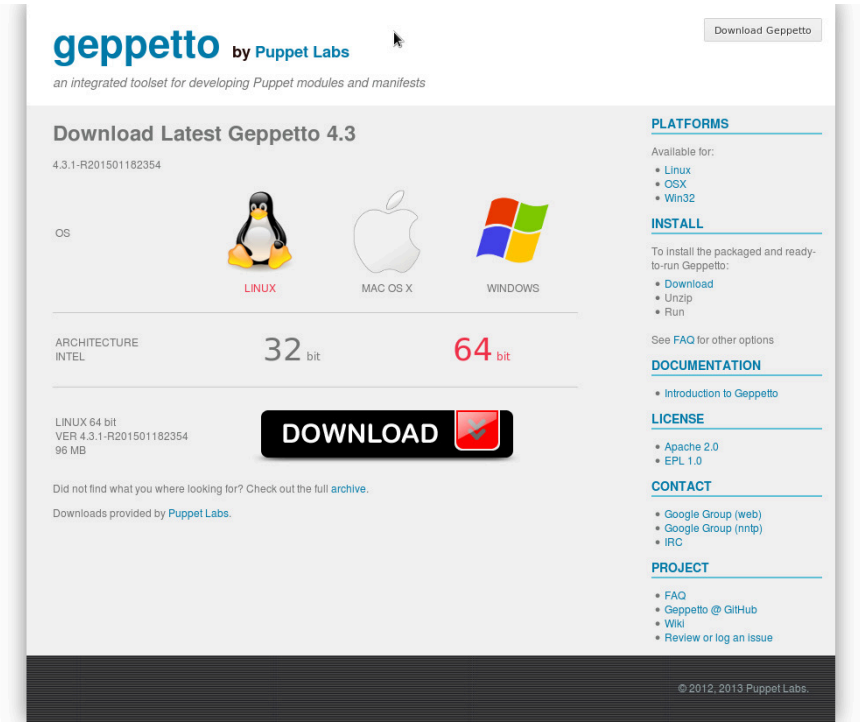

Figure 3.1: Puppet Labs Geppetto software download page

Once the download page appears, the operating system type and architecture can be selected. Available operating systems include Linux, Macintosh OS X, and Windows, and either 32-bit or

64-bit can be selected for the system architecture. After the selections are made, clicking the **Download** button starts the download of a **.zip** archive that contains Geppetto with all of its necessary components.

The **.zip** archive should be extracted into the directory of the user who will develop Puppet content. The archive will create a **geppetto** subdirectory and populate it with over 100 MB of content. The subdirectory will contain an executable called **geppetto**. Executing that script will launch the Geppetto IDE.

Adding Geppetto functionality to Eclipse

The other method for installing Geppetto is to add its functionality to an existing Eclipse environment. After launching Eclipse, select **Help** > **Install New Software**. An **Available Software** dialog box will appear. Locate the **Work with** box and enter the following link: **https://geppetto-updates.puppetlabs.com/4.x**. Expand the **Geppetto** entry that appears in the list of available software, then check the box for installing Geppetto in Eclipse. Click the **Next** button, then read and accept the license agreement. Click **Finish** when the installation is done. Eclipse must be restarted before the Geppetto functionality will be available in the IDE.

Getting started with Geppetto

Geppetto is launched by running the **geppetto** script. This script is found in the **~/geppetto** directory, which was created by the standalone installer.

```
[student@workstation ~]$ ~/geppetto/geppetto
```

When Geppetto launches, it displays the **Workspace Launcher** dialog box. Here the tool's workspace can be specified. The default choice presented is **~/workspace**. The **Browse...** button can be used to select another location, if that is desired. There is a checkbox that prevents this dialog from appearing in future launches. Clicking the **OK** button accepts the selection and causes Geppetto to continue.

A Geppetto project is usually all of the files that makes up a Puppet module. Geppetto can also create and manage projects that are simple manifests. To create a Geppetto project, select **File** > **New** > **Project...**. The **New Project** dialog box will appear. Expand the **Puppet** folder, then select either **Puppet Module Project** to start building a Puppet module or **Puppet Project** for a directory for creating manifests. Click the **Next** button to advance to the dialog for specifying the name of the project being created. Enter the name in the **Project name** field, then click the **Finish** button to create the new project.

The new project being created will be selected in the **Project Explorer** pane at the left side of the screen. To create a new manifest, click the new icon below the **File** dropdown menu. Expand the **Puppet** folder, then select **New Puppet Manifest** from the list of file types. Click the **Next** button, then specify the file name of the manifest that is to be created. Click the **Finish** button to create the new project. The new file will be created and displayed in the upper-right pane.

At this point, Puppet DSL can be typed into the editor. Geppetto will help with the creating of the Puppet manifest by supplying matching punctuation, closed braces, and quotes. The editor will also highlight Puppet manifest syntax by colorizing different elements of resource definitions. Syntax errors and warnings will appear at the left edge of the frame when they affect content being typed. Hovering over the symbols will cause a balloon to appear with a more descriptive message.

Geppetto can also format the Puppet DSL content so it is easy to read. This is accomplished by selecting the content that is to be formatted. Typing **Ctrl**+**A** is a shortcut for selecting all

of the text in the editor. To format the selected text, type **Shift**+**Ctrl**+**F** or right-click the selection, then select **Source** > **Format** from the menu that appears. This is a very useful feature of Geppetto.

Once the manifest has been typed, changes can be saved by clicking on the **Save** icon below the **Edit** pulldown menu. Typing **Ctrl**+**S**, or by selecting **File** > **Save**, are two alternative ways of saving changes to the new Puppet manifest.

References

For more information, see

> Puppet Labs - Introduction to Geppetto
> http://docs.puppetlabs.com/geppetto/4.0/

Guided Exercise: Installing and Using Geppetto

In this exercise, you will install the Geppetto integrated development environment for Puppet and use it to create Puppet DSL.

Resources	
Application URL	**http://content.example.com/puppet3.6/x86_64/dvd/geppetto/geppetto-linux.gtk.x86_64-4.3.1-R201501182354.zip**
Machines	**workstation**

Outcome(s)
You should be able to install Geppetto standalone software on a Linux host and launch it to create and edit Puppet manifests.

Before you begin
You should have a Linux workstation installed with a graphical desktop environment.

1. Log into **workstation** graphically as **student**, with a password of **student**. Download the zip archive that contains the Geppetto IDE from **classroom**.

```
[student@workstation ~]$ curl -O http://content.example.com/puppet3.6/x86_64/dvd/
geppetto/geppetto-linux.gtk.x86_64-4.3.1-R201501182354.zip
```

2. Extract the zip archive to install the Geppetto IDE for use by **student**.

```
[student@workstation ~]$ unzip geppetto-linux.gtk.*.zip
Archive:  geppetto-linux.gtk.x86_64-4.3.1-R201501182354.zip
   creating: geppetto/
  inflating: geppetto/notice.html
  inflating: geppetto/geppetto
  inflating: geppetto/artifacts.xml
  inflating: geppetto/icon.xpm
  inflating: geppetto/epl-v10.html
   creating: geppetto/p2/
... Output omitted ...
  inflating: geppetto/geppetto.ini
```

3. Launch the Geppetto IDE and use **/home/student/workspace** as the Puppet workspace directory.

   3.1. Launch the Geppetto IDE.

```
[student@workstation ~]$ ~/geppetto/geppetto
```

   3.2. Select the default location, **/home/student/workspace**, when the **Workspace Launcher** dialog box opens. Click the **OK** button to continue.

4. Create a new Puppet project called "geppetto-practice". This project will include a Puppet manifest, but not be a Puppet module.

4.1. Select **File** > **New** > **Project...**.

4.2. When the **New Project** dialog box appears, expand the **Puppet** folder, then select **Puppet Project** to create a directory for writing manifests. Click the **Next** button.

4.3. Enter "geppetto-practice" in the **Project name** field, then click **Finish** to create the project.

5. Create a manifest called **`practice.pp`**.

5.1. Click the new wizard icon, below the **File** dropdown menu.

5.2. Expand the **Puppet** folder and select **New Puppet Manifest** from the list of choices presented. Click the **Next** button.

5.3. Enter **`practice.pp`** in the **File name** field, then click the **Finish** button to create an empty manifest.

6. Use the Geppetto editor to add the following content to the manifest:

```
notify { 'hello-world':
  message => 'Hello world!',
}

user { 'webmaster':
  ensure => 'present',
  home => '/home/webmaster',
  shell => '/bin/bash',
}

file { '/home/webmaster':
  ensure => 'directory',
  owner => 'webmaster',
  mode => '770',
  require => User['webmaster'],
}

package { 'httpd':
  ensure => 'present',
}

service { 'httpd':
  ensure => 'running',
  enable => true,
  require => Package['httpd'],
}
```

Note how Geppetto provides matching quotes and braces when you type strings and stanzas into the manifest. Also observe how Geppetto color codes the different elements of the resources after you type them in.

7. When you finish entering the resources into the manifest, reformat the content so it is easy to read. This is accomplished by selecting all of the content in the Geppetto editor, **`Ctrl`**+**A**. Either type **`Shift`**+**`Ctrl`**+**F** or right-click the selection, then select **Source** > **Format** from the menu that appears.

8. Save your changes by clicking the **Save** icon below the **Edit** pulldown menu, by typing **Ctrl**+**S**, or by selecting **File** > **Save**.

9. Select **File** > **Exit** to exit Geppetto.

10. Confirm **practice.pp** contains the nicely formatted Puppet manifest that you just created with Geppetto:

```
[student@workstation ~]$ cat ~/workspace/geppetto-practice/practice.pp
... Output omitted ...
```

Lab: Implementing a Puppet Manifest

In this lab, you will create a Puppet manifest that deploys an FTP server. It will install the *vsftpd* RPM package; create an administrative user, called **ftpadmin**; assign file access permissions to files in **/var/ftp**; and deploy the **vsftpd** service.

Resources	
Files	**/root/ftp-setup.pp**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet manifest that installs a package, creates an administrative user, adjusts directory permissions and ownership, and enables a network service. You should also be able to diagnose the syntax errors in an errant Puppet manifest.

Before you begin
The *puppet* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-manifest setup
```

2. Log in as **root** on **servera** and create a Puppet manifest called **/root/ftp-setup.pp**.

```
[root@servera ~]# vi /root/ftp-setup.pp
```

3. Add Puppet DSL to the manifest that will install the *vsftpd* RPM.

4. Modify the manifest so that it will create an administrative user, called **ftpadmin**. The user's home directory should be the top-level FTP directory, **/var/ftp**. Their shell should be **/bin/bash**.

5. Add Puppet DSL to the manifest that assigns file access permissions to **/var/ftp** and **/var/ftp/pub**. The owner of those directories should be **ftpadmin** and they should have **drwxr-xr-x** permissions.

6. Add Puppet DSL to the manifest to start the **vsftpd** service and configure it to start at boot time.

7. Use the **puppet apply** command to make the state of **servera** conform to the specifications in the manifest.

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-manifest grade
```

Solution

In this lab, you will create a Puppet manifest that deploys an FTP server. It will install the *vsftpd* RPM package; create an administrative user, called **ftpadmin**; assign file access permissions to files in **/var/ftp**; and deploy the **vsftpd** service.

Resources	
Files	**/root/ftp-setup.pp**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet manifest that installs a package, creates an administrative user, adjusts directory permissions and ownership, and enables a network service. You should also be able to diagnose the syntax errors in an errant Puppet manifest.

Before you begin
The *puppet* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-manifest setup
```

2. Log in as **root** on **servera** and create a Puppet manifest called **/root/ftp-setup.pp**.

```
[root@servera ~]# vi /root/ftp-setup.pp
```

3. Add Puppet DSL to the manifest that will install the *vsftpd* RPM.

   Add a **package** resource to **/root/ftp-setup.pp**:

```
package { 'vsftpd':
  ensure  => 'present',
}
```

4. Modify the manifest so that it will create an administrative user, called **ftpadmin**. The user's home directory should be the top-level FTP directory, **/var/ftp**. Their shell should be **/bin/bash**.

   Add a **user** resource to **/root/ftp-setup.pp**:

```
user { 'ftpadmin':
  ensure  => 'present',
  home    => '/var/ftp',
  shell   => '/bin/bash',
  require => Package['vsftpd'],
}
```

   The **require** attribute is needed because the **/var/ftp** directory is created by that package.

5. Add Puppet DSL to the manifest that assigns file access permissions to **/var/ftp** and **/var/ftp/pub**. The owner of those directories should be **ftpadmin** and they should have **drwxr-xr-x** permissions.

   Add the following two **file** resources to **/root/ftp-setup.pp**:

```
file { '/var/ftp':
  ensure  => 'directory',
  owner   => 'ftpadmin',
  mode    => '755',
  require => User['ftpadmin'],
}

file { '/var/ftp/pub':
  ensure  => 'directory',
  owner   => 'ftpadmin',
  mode    => '755',
}
```

   The top-level directory, **/var/ftp**, depends upon the **ftpadmin** user because ownership will be assigned to them.

6. Add Puppet DSL to the manifest to start the **vsftpd** service and configure it to start at boot time.

   Add the following **service** resource to **/root/ftp-setup.pp**:

```
service { 'vsftpd':
  ensure  => 'running',
  enable  => true,
  require => Package['vsftpd'],
}
```

7. Use the **puppet apply** command to make the state of **servera** conform to the specifications in the manifest.

```
[root@servera ~]# rpm -q vsftpd
package vsftpd is not installed
[root@servera ~]# id ftpadmin
id: ftpadmin: no such user
[root@servera ~]# puppet apply ftp-setup.pp
Notice: Compiled catalog for servera.lab.example.com in environment production in
 1.10 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Package[vsftpd]/ensure: created
Notice: /Stage[main]/Main/Service[vsftpd]/ensure: ensure changed 'stopped' to
 'running'
Notice: /Stage[main]/Main/User[ftpadmin]/ensure: created
Notice: /Stage[main]/Main/File[/var/ftp]/owner: owner changed 'root' to 'ftpadmin'
Notice: /Stage[main]/Main/File[/var/ftp/pub]/owner: owner changed 'root' to
 'ftpadmin'
Notice: Finished catalog run in 3.57 seconds
[root@servera ~]# rpm -q vsftpd
vsftpd-3.0.2-9.el7.x86_64
[root@servera ~]# id ftpadmin
uid=1001(ftpadmin) gid=1001(ftpadmin) groups=1001(ftpadmin)
[root@servera ~]# ls -ld /var/ftp /var/ftp/pub
drwxr-xr-x. 3 ftpadmin root 16 Aug  6 02:57 /var/ftp
```

```
drwxr-xr-x. 2 ftpadmin root  6 Mar  7  2014 /var/ftp/pub
[root@servera ~]# systemctl is-active vsftpd
active
[root@servera ~]# systemctl is-enabled vsftpd
enabled
```

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-manifest grade
```

Summary

In this chapter, you learned:

- The required syntax for specifying **`file`**, **`package`**, **`service`**, and **`user`** resources in Puppet DSL.
- The *puppet* package provides Puppet functionality on Red Hat Enterprise Linux.
- The **`puppet apply`** command makes a host conform to the resources specified in the Puppet manifest passed as an argument.

CHAPTER 4

TROUBLESHOOTING PUPPET MANIFESTS

Overview	
Goal	Find documentation and diagnose errors in Puppet manifests.
Objectives	• Locate documentation for Puppet DSL. • Diagnose issues with Puppet manifests.
Sections	• Finding Documentation for Puppet DSL (and Quiz) • Troubleshooting Puppet Manifests (and Guided Exercise)
Lab	• Troubleshooting Puppet Manifests

Finding Documentation for Puppet DSL

Objectives

After completing this section, students should be able to:

- Locate documentation for Puppet DSL.

Using local Puppet documentation

Where would an administrator go to find more information about the Puppet resources and resource parameters that are available? When Puppet is installed on a Red Hat Enterprise Linux system, documentation is installed on the system with it. Use the **`puppet resource --types`** command to see a list of the available Puppet resource types.

```
[root@host ~]# puppet resource --types
augeas
computer
cron
exec
file
filebucket
group
host
interface
... Output omitted ...
```

Listing the Puppet resource types only provides the names of resources. To see the details for using a resource, either generate the local documentation or use the **`puppet describe`** command.

Generating Puppet documentation

The **`puppet doc`** command will generate a reference for all Puppet types. By default, this will send over 6,000 lines of documentation to the screen. Use the **`less`** command to search the documentation for specific resources or features. Each resource section begins with a "###" followed by the name of the resource. The following example used **`/### yumrepo`** to jump to the **`yumrepo`** resource definition.

```
[root@host ~]# puppet doc | less
### yumrepo

The client-side description of a yum repository. Repository
configurations are found by parsing `/etc/yum.conf` and
the files indicated by the `reposdir` option in that file
(see `yum.conf(5)` for details).

Most parameters are identical to the ones documented
in the `yum.conf(5)` man page.

Continuation lines that yum supports (for the `baseurl`, for example)
are not supported. This type does not attempt to read or verify the
existence of files listed in the `include` attribute.

#### Parameters
```

```
baseurl
: The URL for this repository. Set this to `absent` to remove it from the file
 completely.

  Valid values are `absent`. Values can match `/.*/`.
```

In the example there is a general description of the resource followed by a **Parameters** section. Some resources also have a **Features** section before the **Parameters**. Not all features are supported by all providers.

Describing a resource

Another way to view the features of a resource is with the **puppet describe** command.

```
[root@host ~]# puppet describe yumrepo
yumrepo
=======
The client-side description of a yum repository. Repository
configurations are found by parsing `/etc/yum.conf` and
the files indicated by the `reposdir` option in that file
(see `yum.conf(5)` for details).
Most parameters are identical to the ones documented
in the `yum.conf(5)` man page.
Continuation lines that yum supports (for the `baseurl`, for example)
are not supported. This type does not attempt to read or verify the
exinstence of files listed in the `include` attribute.


Parameters
----------

- **baseurl**
    The URL for this repository. Set this to `absent` to remove it from the
    file completely.

... Output omitted ...
```

Unlike the **puppet doc** command, **puppet describe** displays the documentation of a single Puppet resource type that is specified as an argument. The output is more readable, so it can help a Puppet user find more information about resource parameters and possible values when they already know which resource type they are interested in.

Using Puppet Labs reference documentation

Additional documentation can be found on the Puppet Labs website at *https://docs.puppetlabs.com/*. For resource descriptions, look under the **Open Source Puppet** column and select the **Type Reference** link. Hyperlinks will appear for each of the available resource types.

Note

Be aware that the documentation for the latest community version will initially display. Use the links on the left navigation area to select other versions.

The Puppet Labs documentation pages are well organized. They provide reference information and tutorials on a variety of Puppet topics.

References

puppet-doc(8) and **puppet-describe**(8) man pages

For more information, see
https://docs.puppetlabs.com/puppet/#main-docs/
http://docs.puppetlabs.com/references/3.6.0/type.html

For more information, see
Puppet http://docs.puppetlabs.com/puppet/3.6/reference/
http://docs.puppetlabs.com/puppet/3.6/reference/index.html

Quiz: Finding Documentation for Puppet DSL

Use the **puppet doc** and **puppet describe** commands on **servera** to answer the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which one of the following is ***not*** a valid Puppet resource type?

   a. **computer**
   b. **interface**
   c. **manager**
   d. **selboolean**
   e. **tidy**

2. Which one of the following Puppet resources manages host SSH keys?

   a. **computer**
   b. **host**
   c. **service**
   d. **ssh_authorized_key**
   e. **sshkey**

3. Which one of the following is ***not*** a valid parameter for the **package** resource?

   a. **ensure**
   b. **install_options**
   c. **name**
   d. **path**
   e. **source**

4. Which one of the following is ***not*** a possible value for the **ensure** parameter in a **file** resource?

   a. **absent**
   b. **directory**
   c. **file**
   d. **present**
   e. **symlink**

5. According to Puppet documentation, when using a **user** resource to define a local system user account on a host, the user ID assigned to the user will be less than or equal to what value?

   a. A value dependent upon the operating system.
   b. 500
   c. 1000

d. 1024

Solution

Use the **puppet doc** and **puppet describe** commands on **servera** to answer the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Which one of the following is ***not*** a valid Puppet resource type?

   a. computer
   b. interface
   c. manager
   d. selboolean
   e. tidy

2. Which one of the following Puppet resources manages host SSH keys?

   a. computer
   b. host
   c. service
   d. ssh_authorized_key
   e. sshkey

3. Which one of the following is ***not*** a valid parameter for the **package** resource?

   a. ensure
   b. install_options
   c. name
   d. path
   e. source

4. Which one of the following is ***not*** a possible value for the **ensure** parameter in a **file** resource?

   a. absent
   b. directory
   c. file
   d. present
   e. symlink

5. According to Puppet documentation, when using a **user** resource to define a local system user account on a host, the user ID assigned to the user will be less than or equal to what value?

   a. A value dependent upon the operating system.
   b. 500
   c. 1000

d. 1024

Troubleshooting Puppet Manifests

Objectives

After completing this section, students should be able to:

- Diagnose issues with Puppet manifests.

Checking syntax with `puppet parser validate`

It is common for new Puppet users to make simple syntax errors when they create their manifests. The Puppet parser can be invoked from the command line with the **`puppet parser validate`** command. The argument to the command is the file name of the manifest to check. The following is a working Puppet manifest without any syntax errors.

```
[root@host ~]# cat -n sample.pp
     1  yumrepo { 'rhel7osp':
     2    baseurl  => 'http://content.example.com/puppet3.6/x86_64/dvd/rhel7osp/',
     3    enabled  => '1',
     4    gpgcheck => '0',
     5    descr    => 'Red Hat Enterprise Linux OpenStack Platform 7',
     6  }
     7
     8  package { 'openstack-puppet-modules':
     9    ensure  => 'present',
    10    require => Yumrepo['rhel7osp'],
    11  }
```

`puppet parser validate` does not generate any output when there are no syntax errors to report. It also exits with a zero exit status in this case so it can be used in shell programs.

```
[root@host ~]# puppet parser validate sample.pp
[root@host ~]# echo $?
0
```

In the case of a syntax error, the parser will return an error with information concerning the bad syntax. The following Puppet DSL snippet comes from a manifest that is missing a brace at line 6.

```
     4   gpgcheck => '0',
     5   descr => 'Red Hat Enterprise Linux OpenStack Platform 7',
     6
     7
     8  package { 'openstack-puppet-modules':
     9   ensure => 'present',
```

When a brace is missing, the **`puppet parser validate`** command produces an error referencing a line further in the manifest and produces a "expected '}'" message. The error occurs when the parser reaches an item that should not be inside the open section, which may be one or more lines after the missing brace.

```
[root@host ~]# puppet parser validate sample.pp
Error: Could not parse for environment production: Syntax error at '{';
 expected '}' at /root/sample.pp:8
```

Other typographical errors cause similar errors in both the line location and the brace reference. The following manifest is missing a comma separating resource parameters at line 3.

```
[root@host ~]# cat -n sample.pp
     1  yumrepo { 'rhel7osp':
     2    baseurl  => 'http://content.example.com/puppet3.6/x86_64/dvd/rhel7osp/',
     3    enabled  => '1'
     4    gpgcheck => '0',
     5    descr    => 'Red Hat Enterprise Linux OpenStack Platform 7',
     6  }
[root@host ~]# puppet parser validate sample.pp
Error: Could not parse for environment production: Syntax error at 'gpgcheck';
 expected '}' at /root/sample.pp:4
```

Again, the parser produces an error referencing the next line and expecting a closing brace. The Puppet parser thinks **gpgcheck** might be a new resource block.

Troubleshooting with `puppet apply --noop`

Some Puppet manifest errors have correct syntax so **`puppet parser validate`** does not detect them and produce output. Any resource parameter can be given, but the parser will not know if it is correct or valid until the manifest is applied.

The following Puppet manifest has a misspelled parameter name on line 4.

```
[root@host ~]# cat -n sample.pp
     1  yumrepo { 'rhel7osp':
     2    baseurl  => 'http://content.example.com/puppet3.6/x86_64/dvd/rhel7osp/',
     3    enabled  => '1',
     4    pgcheck => '0',
     5    descr    => 'Red Hat Enterprise Linux OpenStack Platform 7',
     6  }
```

The syntax for the manifest is correct. All commas, braces, quotes, and equal signs are in their correct locations. Validating the manifest with **`puppet parser validate`** returns an exit status of 0.

```
[root@host ~]# puppet parser validate sample.pp
[root@host ~]# echo $?
0
```

Before applying a manifest to a system, use **`puppet apply --noop`** to perform a test run without making any changes to the system. This command takes the additional step of interpreting resource parameters and checking their assigned values.

```
[root@host ~]# puppet apply --noop sample.pp
Error: Invalid parameter pgcheck on Yumrepo[rhel7osp] at /root/sample.pp:6
 on node host.example.com
Wrapped exception:
Invalid parameter pgcheck
Error: Invalid parameter pgcheck on Yumrepo[rhel7osp] at /root/sample.pp:6
 on node host.example.com
```

The resulting error message points to an invalid parameter. Use **`puppet describe yumrepo`** to verify the correct and valid parameter names for that resource. Once the manifest has been fixed and the **`--noop`** test passes without error, the manifest can be applied to the system.

References

`puppet-parser`(8) and **`puppet-apply`**(8) man pages

Guided Exercise: Troubleshooting Puppet Manifests

In this lab, you will diagnose and correct syntax errors in Puppet manifests that have errors. You will be presented with five broken manifests. Each of them will have a single error that must be corrected. Use your Puppet troubleshooting skills to identify the error, fix it, and confirm that the problem has been solved.

Resources	
Files	**broken1.pp**, **broken2.pp**, **broken3.pp**, **broken4.pp**, and **broken5.pp**
Machines	**servera**

Outcome(s)
You should be able to diagnose issues with Puppet manifests.

Before you begin
The *puppet* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script. It will publish the broken Puppet manifests to **/root** on **servera**.

```
[student@workstation ~]$ lab puppet-ts-practice setup
```

2. Use **puppet parser validate** to parse the **broken1.pp** manifest. Examine the error message to help you identify the problem.

```
[root@servera ~]# puppet parser validate broken1.pp
Error: Could not parse for environment production: Syntax error at
 'shell'; expected '}' at /root/broken1.pp:14
```

Edit **broken1.pp** and look at the Puppet DSL around line 14. Line 13 is missing a separating comma, so add it.

```
    11  user { 'webmaster':
    12    ensure => 'present',
    13    home   => '/home/webmaster',
    14    shell  => '/bin/bash',
    15  }
```

Execute **puppet parser validate** again to confirm the syntax error has been fixed.

```
[root@servera ~]# puppet parser validate broken1.pp
```

3. Use **puppet parser validate** to parse the **broken2.pp** manifest. Examine the error message to help you identify the problem.

```
[root@servera ~]# puppet parser validate broken2.pp
```

```
Error: Could not parse for environment production: Syntax error at '{';
 expected '}' at /root/broken2.pp:18
```

Edit **broken2.pp** and look at the Puppet DSL surrounding line 18. Line 16 is blank. It should have a closing brace: **}**.

```
    11  file { '/home/webmaster':
    12    ensure  => 'directory',
    13    owner   => 'webmaster',
    14    mode    => '0700',
    15    require => User['webmaster'],
    16  }
    17
    18  package { 'httpd':
```

Make the correction and execute **puppet parser validate** to confirm the syntax error has been fixed.

4. Use **puppet parser validate** to parse the **broken3.pp** manifest. Examine the error message to help you identify the problem.

```
[root@servera ~]# puppet parser validate broken3.pp
Error: Could not parse for environment production: Syntax error at 't';
 expected '}' at /root/broken3.pp:2
```

Edit **broken3.pp** and look at the Puppet DSL on line 2. There is an apostrophe enclosed within the single quotes. Enclose the message value in double quotes and save your change.

```
     1  notify { 'hello-world':
     2    message => "Something isn't right about this message!",
     3  }
```

Make the correction and execute **puppet parser validate** to confirm the syntax error has been fixed.

5. Use **puppet parser validate** to parse the **broken4.pp** manifest. Examine the error message to help you identify the problem.

```
[root@servera ~]# puppet parser validate broken4.pp
Error: Could not parse for environment production: Syntax error at '->';
 expected '}' at /root/broken4.pp:13
```

Edit **broken4.pp**. The error is found on line 13. Change the **->** to **=>** in the resource parameter definition.

```
    11  file { '/home/webmaster':
    12    ensure  => 'directory',
    13    owner   => 'webmaster',
    14    mode    => '0700',
    15    require => User['webmaster'],
    16  }
```

Make the correction and execute **puppet parser validate** to confirm the syntax error has been fixed.

6. Use **puppet parser validate** to parse the **broken5.pp** manifest.

```
[root@servera ~]# puppet parser validate broken5.pp
```

This manifest does not have a syntax error that the parser caught. Use the **puppet apply --noop** command to perform additional checking on the manifest. Examine the error messages to help you identify the problem.

```
[root@servera ~]# puppet apply --noop broken5.pp
Error: Invalid parameter snell on User[webmaster] at /root/broken5.pp:9
 on node servera.lab.example.com
Wrapped exception:
Invalid parameter snell
Error: Invalid parameter snell on User[webmaster] at /root/broken5.pp:9
 on node servera.lab.example.com
```

Edit **broken5.pp**. The parameter on line 8 is misspelled. It is supposed to be **shell**, not **snell**. Make the correction and save your change.

```
     5  user { 'webmaster':
     6    ensure => 'present',
     7    home   => '/home/webmaster',
     8    shell  => '/bin/bash',
     9  }
```

Running **puppet apply --noop** after making the correction should produce no errors.

```
[root@servera ~]# puppet apply --noop broken5.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.08 seconds
... Output omitted ...
Notice: Finished catalog run in 0.61 seconds
```

Lab: Troubleshooting Puppet Manifests

In this lab, you will correct syntax errors in a broken Puppet manifest.

Resources	
Files	**broken-lab.pp**
Machines	**servera**

Outcome(s)
You should be able to diagnose and correct errors in Puppet manifests.

Before you begin
The *puppet* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script to create a broken manifest: **broken-lab.pp**. This manifest will be copied to root's home directory on **servera**.

```
[student@workstation ~]$ lab puppet-ts-lab setup
```

2. Use the Puppet parser to check the **broken-lab.pp** manifest for syntax errors. Correct the syntax errors that it identifies.

3. Use the **puppet apply --noop** command on **broken-lab.pp** to identify additional errors. Correct any errors that it identifies.

4. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-ts-lab grade
```

Solution

In this lab, you will correct syntax errors in a broken Puppet manifest.

Resources	
Files	**broken-lab.pp**
Machines	**servera**

Outcome(s)
You should be able to diagnose and correct errors in Puppet manifests.

Before you begin
The *puppet* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script to create a broken manifest: **broken-lab.pp**. This manifest will be copied to root's home directory on **servera**.

```
[student@workstation ~]$ lab puppet-ts-lab setup
```

2. Use the Puppet parser to check the **broken-lab.pp** manifest for syntax errors. Correct the syntax errors that it identifies.

   2.1. Use **puppet parser validate** to check the syntax of **broken-lab.pp**.

```
[root@servera ~]# puppet parser validate broken-lab.pp
Error: Could not parse for environment production: Syntax error at 'require';
 expected '}' at /root/broken-lab.pp:9
```

   It identified an error on line 9, specifically by the string 'require'.

   2.2. Use **cat -n** to display the contents of the manifest with line numbers. This will help you identify the line with the error.

```
[root@servera ~]# cat -n broken-lab.pp
     1  notify { 'webmaster_note':
     2    message => "creating webmaster's account and files",
     3  }
     4
     5  user { 'webmaster':
     6    ensure  => 'present',
     7    home    => '/var/www',
     8    shell   => '/bin/bash'
     9    require => Package['httpd-server'],
    10  }
    11
    12  file { '/var/www':
    13    ensure  => 'directory',
    14    owner   => 'webmaster',
    15    mode    -> '0755',
    16    require => User['webmaster'],
    17  }
    18
    19  package { 'httpd':
    20    ensure  => 'present',
```

```
21    path    => '/usr/local',
22  }
23
24  service { 'httpd':
25    ensure  => 'installed',
26    enable  => yes,
27    require => Package['httpd'],
28  }
```

2.3. The error is actually on line 8. The **shell** parameter is missing a comma separator. Modify **broken-lab.pp** and add a comma to the end of line 8.

```
 7    home    => '/var/www',
 8    shell   => '/bin/bash',
 9    require => Package['httpd-server'],
```

2.4. Recheck the manifest with **puppet parser validate**. If the previous syntax error has been corrected, it will display a different error.

```
[root@servera ~]# puppet parser validate broken-lab.pp
Error: Could not parse for environment production: Syntax error at '->';
 expected '}' at /root/broken-lab.pp:15
```

It identified an error on line 15, specifically by '->'.

2.5. The operator used to assign parameters in resource definitions should be **=>**, not **->**. Correct the error on line 15.

```
14    owner   => 'webmaster',
15    mode    => '0755',
16    require => User['webmaster'],
```

2.6. Recheck the manifest with **puppet parser validate**. If the previous error has been corrected, it will not display any output.

```
[root@servera ~]# puppet parser validate broken-lab.pp
```

3. Use the **puppet apply --noop** command on **broken-lab.pp** to identify additional errors. Correct any errors that it identifies.

3.1.

```
[root@servera ~]# puppet apply --noop broken-lab.pp
Error: Invalid parameter path on Package[httpd] at /root/broken-lab.pp:22
 on node servera.lab.example.com
Wrapped exception:
Invalid parameter path
Error: Invalid parameter path on Package[httpd] at /root/broken-lab.pp:22
 on node servera.lab.example.com
```

It identified an error on line 22. "Invalid parameter path" is the key to solving this error.

```
20    ensure  => 'present',
21    path    => '/usr/local',
22  }
```

3.2. Use the **puppet describe** command to check the possible parameters for the **package** resource.

```
[root@servera ~]# puppet describe package -s

package
=======
Manage packages.  There is a basic dichotomy in package
... Output omitted ...

Parameters
----------
    adminfile, allow_virtual, allowcdrom, category, configfiles,
    description, ensure, flavor, install_options, instance, name,
    package_settings, platform, responsefile, root, source, status,
    uninstall_options, vendor

... Output omitted ...
```

path is not a valid parameter name for the **package** resource. Edit the manifest and comment line 21.

```
    20    ensure  => 'present',
    21  # path    => '/usr/local',
    22  }
```

3.3. Recheck the manifest with **puppet apply --noop**. If the previous error has been corrected, it will display a different error message.

```
[root@servera ~]# puppet apply --noop broken-lab.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.55 seconds
... Output omitted ...
Error: Parameter ensure failed on Service[httpd]: Invalid value
 "installed". Valid values are stopped, running.  at /root/broken-lab.pp:28
Wrapped exception:
Invalid value "installed". Valid values are stopped, running.
Wrapped exception:
Invalid value "installed". Valid values are stopped, running.
```

3.4. It identified the error on line 28, something to do with an invalid value "installed". Looking at the resource definition at the end of the file, the **ensure** parameter is being assigned the value **installed**.

```
    24  service { 'httpd':
    25    ensure  => 'installed',
    26    enable  => yes,
    27    require => Package['httpd'],
    28  }
```

3.5. Use the **puppet describe** command to check the possible values for the **ensure** parameter of the **service** resource.

```
[root@servera ~]# puppet describe service | less
```

Searching for the **ensure** string, you come upon the following paragraph in the documentation.

```
- **ensure**
    Whether a service should be running.
    Valid values are `stopped` (also called `false`), `running` (also called
    `true`).
```

installed is not a valid value for the **ensure** parameter of the **service** resource. Change the value to either **running** or **true**.

```
    24  service { 'httpd':
    25    ensure  => 'running',
    26    enable  => yes,
```

3.6. Recheck the manifest with **puppet apply --noop**. If the previous error has been corrected, it will display a different error message.

```
[root@servera ~]# puppet apply --noop broken-lab.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.56 seconds
... Output omitted ...
Error: Could not find dependency Package[httpd-server] for User[webmaster]
 at /root/broken-lab.pp:10
```

3.7. Puppet identified an error on line 10, something to do with a dependency that could not be found.

```
     8    shell   => '/bin/bash'
     9    require => Package['httpd-server'],
    10  }
```

Line 9 refers to a **package** resource with **httpd-server** as the title. Later in the file, a **package** resource is defined with a title of **httpd**.

```
    18
    19  package { 'httpd':
    20    ensure  => 'present',
```

It appears that the reference has been mistyped. Edit the file and change the reference on line 9 to look like the following.

```
     8    shell   => '/bin/bash'
     9    require => Package['httpd'],
    10  }
```

3.8. Recheck the manifest with **puppet apply --noop**. If the syntax error has been corrected, it will not display any **Error:** messages.

```
[root@servera ~]# puppet apply --noop broken-lab.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.57 seconds
```

```
... Output omitted ...
Notice: /Stage[main]/Main/Notify[webmaster_note]/message: current_value
 absent, should be creating webmaster's account and files (noop)
Notice: /Stage[main]/Main/Package[httpd]/ensure: current_value absent,
 should be present (noop)
Notice: /Stage[main]/Main/Service[httpd]/ensure: current_value stopped,
 should be running (noop)
Notice: /Stage[main]/Main/User[webmaster]/ensure: current_value absent,
 should be present (noop)
Notice: /Stage[main]/Main/File[/var/www]/ensure: current_value absent,
 should be directory (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 5 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.31 seconds
```

4. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-ts-lab grade
```

Summary

In this chapter, you learned:

- The **`puppet doc`** command generates Puppet documentation and references, optionally for a manifest.
- The **`puppet describe`** command displays information about the use and parameters of Puppet resources.
- The *Puppet 3.6 Reference Manual* [http://docs.puppetlabs.com/puppet/3.6/reference/index.html] is useful Puppet documentation published by Puppet Labs.
- The **`puppet parser validate`** command can identify simple Puppet DSL syntax errors.
- The **`puppet apply --noop`** command will identify additional errors in a manifest, such as invalid parameters.

CHAPTER 5

IMPLEMENTING GIT

Overview	
Goal	Implement Git to manage software.
Objectives	• Implement a Git repository. • Manage code with Git.
Sections	• Implementing a Git Repository (and Guided Exercise) • Managing Code with Git (and Quiz)
Lab	• Implementing Git

Implementing a Git Repository

Objectives

After completing this section, students should be able to:

- Implement a Git repository.

Introducing Git

Git is a distributed revision control system that allow developers to manage changes to files in a project. Revision control systems have many benefits, including:

- Each change is committed with a log message. This allows the developer to declare why the change was made.
- Files can be rolled back to an earlier version. This is useful if a working file gets broken, then is committed into the repository.
- File changes between commits can be displayed. A history of changes to a file show how it has evolved through the life of the project.
- A revision control system also records who makes a change and the date/time when it was committed. This permits other project members to identify who to blame (or praise) for a given change.
- Multiple contributors can commit changes to a shared project. Revision control systems allow users to merge their changes together so one contributor's changes do not overwrite someone else's changes.
- Many revision control systems, including Git, have hooks that facilitate workflow automation. More about this will be discussed later in this section.

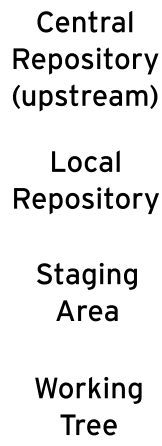

Figure 5.1: The four areas where Git manages files

Git stores its metadata and object database in a *repository*. Users can create a new project or they can clone an existing, shared project from a *central repository*. Git has all of the advantages

of revision control systems previously listed, but what distinguishes it from others is its distributed nature. When a project is cloned from a central repository, the local copy is called the *local repository*. It is a complete copy of the original, upstream repository, not just the latest snapshot of the project files. Git users always interact with the local repository, occasionally pushing changes back to the central repository. This avoids having a single point of failure and users can still manage changes to files even when the network is unavailable.

The *working tree* is a checkout of a single snapshot of a project. This is initially taken from the latest revision of the default **master** branch from the repository. The files of the working tree are available for review and modification.

To use Git effectively, a user must be aware of the three states a file can be in: committed, modified, and staged. A file is *committed* when its data is stored in the local Git repository. The *modified* state means that a file has been added, removed, or changed, but it has not been added to the index of files that will be committed into the repository. A *staged* file is a modified file that has been added to go into the next commit into the repository.

Creating a Git repository

The **git init** command is used to create a Git repository. It is used to create both private repositories and shared repositories. The following command creates a new empty, local repository for a project:

```
[user@host ~]$ git init PROJECT
Initialized empty Git repository in /home/user/PROJECT/.git/
[user@host ~]$ ls -a PROJECT
.  ..  .git
```

An new directory called ***PROJECT*** is created that contains only a **.git** subdirectory. The objects and metadata of the local repository are stored in **.git**. The ***PROJECT*** directory is also a working tree that initially has no files in it. Git repositories also have a staging area.

git init can also place an existing directory under control of Git. The owner of the directory changes to the directory, then executes the following command:

```
[user@host projectdir]$ git init .
```

Git will make the current directory a local repository by creating a **.git** directory beneath it. The existing files will remain untouched in the directory, which serves as a working tree. Initially the repository will be empty. Files will have to be added separately because Git does not make any assumptions about which files should be added to the repository.

The following **git init** command is used to create an empty Git repository that can be shared by multiple users:

```
[user@host projectdir]$ git init --bare --shared=true PROJECT
Initialized empty shared Git repository in /home/user/PROJECT/
[user@host ~]$ ls -a PROJECT
.  ..  branches  config  description  HEAD  hooks  info  objects  refs
```

The **--bare** option creates the repository without a working tree. Instead of keeping the repository metadata in a **.git** subdirectory, all of the Git administration directories are visible in the top-level directory. The **--shared=true** option creates the directory group writeable

with the setgid bit set. This causes subdirectories and files to inherit the group ownership of the directory instead of the users committing changes to the repository.

Automating workflow with Git hooks

Git hooks are programs that run during different workflow stages when Git is executed. They can be shell scripts or binary programs. The only requirement is that they are executable files with specific file names, based on the stage of the Git workflow when they should be executed (i.e., **commit-msg**, **pre-commit**, **post-commit**, and **post-receive**).

Git creates a directory called **hooks** when a new repository is created. A local repository, created by **git init**, creates a **.git/hooks** subdirectory for creating Git hooks. Initially Git populates this directory with example scripts that have a **.sample** filename suffix. The following sample output shows the sample Git hook scripts that are created by default.

```
[user@host project]$ ls .git/hooks/
applypatch-msg.sample  pre-applypatch.sample      pre-push.sample
commit-msg.sample      pre-commit.sample          pre-rebase.sample
post-update.sample     prepare-commit-msg.sample  update.sample
```

The **hooks** directory is located at the top of a shared Git repository that is created with the **git init --bare --shared=true** command. It is initially populated with the same sample scripts.

Automating Puppet syntax checking

Syntax checking is an easy Puppet workflow task that could be automated with Git hooks. When Puppet manifests are under revision control, it would be great if manifests with syntax errors can be detected (and fixed) before they are committed into the repository. The ideal Git hook for this task is the **pre-commit** hook.

The following **pre-commit** hook checks the Puppet syntax of all files that have a **.pp** suffix. This hook executes when staged files are being committed into a repository, before the log message is created or entered. Git does not pass command-line arguments or information to the script via standard input. The **git diff-index** command on line 5 generates a list of all of the staged files that are being committed.

The **for** loop processes each file one at a time. On line 8, each file that ends with a **.pp** suffix is passed as an argument to the **puppet parser validate** command. If the command exits with a nonzero exit status (the manifest has errors), the body of the **if** statement executes. An error message is displayed to the developer on line 10, and a flag variable is set on line 11. The script does not exit immediately at that point, so multiple Puppet manifests can be checked during a single commit.

The Git hook exits on line 14. It uses the **exit_status** variable to determine the exit status of the script. A nonzero exit status causes Git to abort the commit. None of the files are committed until the issues are resolved.

```
 1  #!/bin/bash
 2
 3  exit_status=0
 4  # Validate Puppet syntax of manifests
 5  for file in $(git diff-index --name-only --cached HEAD --)
 6  do
 7    # Puppet manifest ends with .pp, so let Puppet parse it
 8    if [[ ${file} == *.pp ]] && ! puppet parser validate ${file}
 9    then
```

```
    10      echo "Commit failed: Puppet cannot parse ${file}"
    11      exit_status=1
    12    fi
    13  done
    14  exit ${exit_status}
```

After a manifest has been staged, **`git commit`** will generate an error message when a manifest being committed has a syntax error that **`puppet parser validate`** can detect.

```
[user@host project]$ git commit
Error: Could not parse for environment production: Syntax error at '=>';
 expected '}' at /home/user/project/test.pp:5
Commit failed: Puppet cannot parse test.pp
```

References

`git-init`(1), **`githooks`**(5), and **`gittutorial`**(7) man pages

For more information, see
> Git Documentation
> http://git-scm.com/documentation

For more information, see
> drwahl/puppet-git-hooks
> https://github.com/drwahl/puppet-git-hooks

Guided Exercise: Implementing a Git Repository

In this lab, you will create a shared Git repository and create a project for storing Puppet content.

Resources	
Files	**/var/git**
Application URL	**http://materials.example.com/manifests/git-repo.pp**
Machines	**servera**

Outcome(s)
You should be able to configure a shared Git repository.

Before you begin
You should have a Red Hat Enterprise Linux server installed with **yum** configured.

1. Log into **servera** as **root**. Download the Puppet manifest found at **http://materials.example.com/manifests/git-repo.pp**. It will install Git, create a **git** group, and create a shared directory at **/var/git** when it is applied.

```
[root@servera ~]# curl -O http://materials.example.com/manifests/git-repo.pp
... Output omitted ...
[root@servera ~]# cat git-repo.pp
# This Puppet manifest prepares a system to serve as a shared Git repo.
#
# It does the following:
#
#  - Make sure the git package is installed
#  - Create a 'git' system group
#  - Make sure 'student' is a member of the 'git' group
#  - Create a /var/git directory, owned by the 'git' group, perms 2770

group { 'git':
  ensure  => 'present',
  system  => true,
}

user { 'student':
  ensure  => 'present',
  groups  => 'git',
  require => Group['git'],
}

file { '/var/git':
  ensure  => 'directory',
  owner   => 'root',
  group   => 'git',
  mode    => '2770',
  require => Group['git'],
}

package { 'git':
  ensure  => 'present',
```

```
}
```

2. Use Puppet to apply the manifest to create the group and shared directory.

```
[root@servera ~]# puppet apply git-repo.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.51 seconds
Warning: The package type's allow_virtual parameter will be changing its
 default value from false to true in a future release. If you do not want
 to allow virtual packages, please explicitly set allow_virtual to false.
   (at /usr/share/ruby/vendor_ruby/puppet/type.rb:816:in `set_default')
Notice: /Stage[main]/Main/Package[git]/ensure: created
Notice: /Stage[main]/Main/Group[git]/ensure: created
Notice: /Stage[main]/Main/File[/var/git]/ensure: created
Notice: /Stage[main]/Main/User[student]/groups: groups changed 'wheel'
 to 'git,wheel'
Notice: Finished catalog run in 3.96 seconds
```

3. As the **student** user, create an empty, shared Git repository for the **puppet** project.

```
[root@servera ~]# su - student
[student@servera ~]$ cd /var/git
[student@servera git]$ git init --bare --shared=true puppet.git
Initialized empty shared Git repository in /var/git/puppet.git/
[student@servera git]$ ls puppet.git
branches  config  description  HEAD  hooks  info  objects  refs
```

Managing Code with Git

Objectives

After completing this section, students should be able to:

- Manage code with Git.

Since Git users frequently modify projects with multiple contributors, Git publishes the user's name and email address with each commit. These values can be defined at a project level, but global defaults can also be set for a user. The **`git config`** command controls these settings. Using it with the **`--global`** option manages the default settings for all Git projects the user contributes to.

```
[peter@host ~]$ git config --global user.name 'Peter (Starlord) Quill'
[peter@host ~]$ git config --global user.email peter@host.example.com
```

The Git workflow

When working on shared projects, the Git user clones an existing upstream repository with the **`git clone`** command. The pathname or URL provided determines which repository is cloned into the current directory. A working tree is also created so that the directory of files is ready for revisions. Since the working tree is unmodified, it is initially in the clean state.

Note

Another way to start the Git workflow is to create a new, private project with the **`git init`** command.

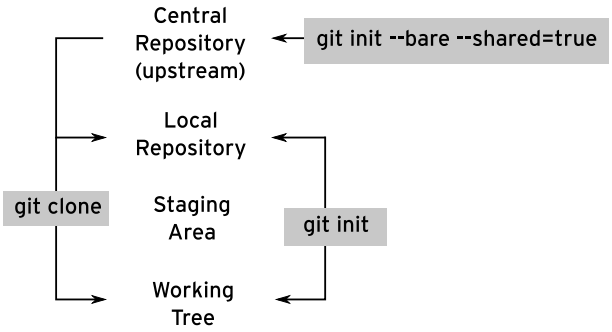

Figure 5.2: Git subcommands used to create a repository

New files are created and existing files are modified in the working tree. This changes the working tree to a dirty state. The **`git status`** command will display information about changed files in the working tree.

The **`git add`** command changes a file from the modified to the staged state. It is included in the staging area that will be saved in the repository at the next commit. This step should be performed when the files are in a stable and usable state within the rest of the project.

Note

The **`git rm`** command removes an unmodified file and adds it to the index of staged files.

The **`git commit`** command commits the changes to the staged files to the local Git repository. A log message must be provided that explains why the current set of staged files is being saved. Log messages do not have to be long, but they must be meaningful to be useful.

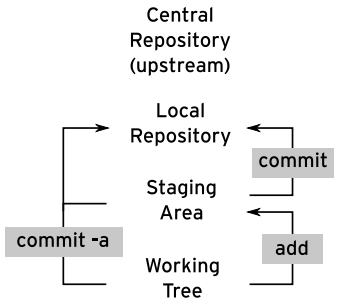

Figure 5.3: Git subcommands that add/update local repository content

The **`git push`** command uploads changes made to the local repository to the original, upstream repository. This is how a Git user publishes changes to the others working on the project. Before Git pushes will work, the default push method must be defined. The following command sets the default push method to the **`simple`** method. This is the safest option for beginners and will be the default push method in Git 2.0.

```
[peter@host ~]$ git config --global push.default simple
```

The **`git pull`** command retrieves updates from the upstream repository and saves them to the local repository. It also updates the files in the working tree. This command should be done frequently to stay current with the changes that others are making to the project. The **`git fetch`** command does the same thing as **`git pull`**, except the working tree is left untouched.

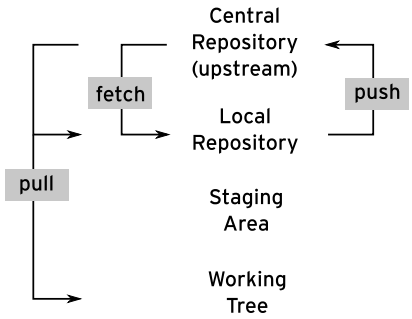

Figure 5.4: Git subcommands that interact with a central repository

During the course of edits, the following Git subcommands will be frequently used to check the status and current state of files in the project: **`status`**, **`diff`**, **`log`**, and **`show`**. **`git status`** displays the general state of modified, staged, and new files outside of Git's control in the

working tree. **`git diff`** displays how a file has been modified in the working tree from its copy in the repository. The **`git log`** command displays the commit log messages with their associated ID hashes for a file. A hash can be used with the **`git show`** command to display the changes to that file that were committed to the repository.

The **`git reset`** command removes a file from the staging area that has been added for a later commit. This command has no effect on the file's contents. The **`git checkout --`** command restores the file contents to the latest state that have been checked into the local repository.

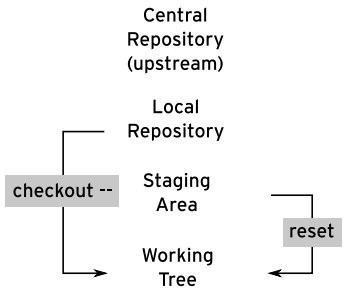

Figure 5.5: Git subcommands that interact with the working tree

Git Quick Reference

Command	Description
`git clone`	Clone an existing Git project into the current directory.
`git status`	Display the status of modified and staged files in a working tree.
`git add`	Stage a file for the next commit.
`git commit`	Commit the staged files with a descriptive message.
`git diff`	Display the differences of files in the working tree with the latest version in the local repository.
`git log`	View a repository's commit history for a file.
`git show`	Show the log message and textual differences when used with a commit ID.
`git push`	Push the local repository to the original upstream Git repository.
`git pull`	Pull updates from the upstream Git repository to the local repository and merge them into the working tree.
`git fetch`	Fetch updates from the upstream Git repository to the local repository without modifying the working tree.
`git checkout --`	Restore modified files in the working tree to their latest repository state.
`git reset`	Unstage a file from the next commit (the opposite of **`git add`**).
`git config`	Manage Git configuration values; for example author name, email address, and default push mode.
`git init`	Create a new Git repository in the current directory.

References

gittutorial(7) man page

For more information, see

> GitHub Getting Started Tutorial
> http://try.github.io

Guided Exercise: Managing Code with Git

Resources	
Files	**manifests/motd.pp** and **README**
Application URL	**ssh://servera/var/git/puppet.git**
Machines	**servera** and **serverb**

Outcome(s)
You should be able to create a Git project, add files to a project, commit changes, and check the status of the project.

Before you begin
You should have a shared Git repository at **/var/git** on **servera**. It should contain a new, empty project called **puppet**. The **student** user should be a Git administrator.

1. Log into **workstation** as **student** and run the lab setup script. This will create a user account named **terry**, with an account password of **tester123**, on the two servers used for the lab.

```
[student@workstation ~]$ lab puppet-git-workshop setup
```

2. Log into **servera** as **student** (password is **student**). Configure the Git name and email address.

```
[student@servera ~]$ git config --global user.name 'Student User'
[student@servera ~]$ git config --global user.email student@servera.lab.example.com
[student@servera ~]$ cat ~/.gitconfig
[user]
        name = Student User
        email = student@servera.lab.example.com
```

Note

You may need to log out and back in as **student** to make sure **git** is one of the active groups. Use the **id** command to confirm this.

```
[student@servera ~]$ id
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel),994(git)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

3. Clone the **puppet.git** project from the shared repository. This will create a repository with a working tree in a subdirectory called **puppet**.

```
[student@servera ~]$ git clone /var/git/puppet.git
Cloning into 'puppet'...
warning: You appear to have cloned an empty repository.
done.
[student@servera ~]$ ls -a puppet
```

```
.  ..  .git
```

Initially, the repository is empty because no files were added to it yet.

4. Change into the working tree and populate it with content. Copy all the files in **/etc/puppet** into the working tree. Use **git status** to display the status of the Git working tree.

```
[student@servera ~]$ cd puppet
[student@servera puppet]$ cp -a /etc/puppet/* .
[student@servera puppet]$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       auth.conf
#       puppet.conf
nothing added to commit but untracked files present (use "git add" to track)
```

Git is aware of **auth.conf** and **puppet.conf**, but they are not managed by Git at this point.

5. Make a new directory called **manifests** and create a file in it called **motd.pp**.

5.1. The file should contain the following content:

```
file { '/etc/motd':
  ensure  => 'file',
  owner   => 'root',
  group   => 'root',
  mode    => '444',
  content => "No trespassing.\n",
}
```

5.2. **git status** displays that Git is aware of the new file.

```
[student@servera puppet]$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       auth.conf
#       manifests/
#       puppet.conf
nothing added to commit but untracked files present (use "git add" to track)
```

6. Stage all the files in the **puppet** directory.

```
[student@servera puppet]$ git add *
[student@servera puppet]$ git status
# On branch master
#
```

```
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   auth.conf
#       new file:   manifests/motd.pp
#       new file:   puppet.conf
#
```

Git displays that the files are now ready to be committed to the repository. Notice that the file under the matched directory was staged.

7. Commit the changes to the local repository with a useful log message.

```
[student@servera puppet]$ git commit -m 'Initial /etc/puppet content'
[master (root-commit) 55c6f71] Initial /etc/puppet content
 3 files changed, 152 insertions(+)
 create mode 100644 auth.conf
 create mode 100644 manifests/motd.pp
 create mode 100644 puppet.conf
```

8. Terry Tester is another user working on the Puppet project. Terry wants to be able to update the project from Terry's computer.

   Open a new terminal window and log into **serverb** as **terry** (password is **tester123**). Configure the Git name and email address.

```
[terry@serverb ~]$ git config --global user.name 'Terry Tester'
[terry@serverb ~]$ git config --global user.email terry@serverb.lab.example.com
```

9. Clone the shared **puppet.git** repository. Note how the repository is specified when it resides on a remote server.

```
[terry@serverb ~]$ git clone ssh://servera/var/git/puppet.git
Cloning into 'puppet'...
The authenticity of host 'servera (172.25.250.10)' can't be established.
ECDSA key fingerprint is 11:92:7e:64:b5:4a:c7:94:1e:ea:c6:62:12:d7:2c:5a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'servera,172.25.250.10' (ECDSA) to the list of known
 hosts.
warning: You appear to have cloned an empty repository.
```

10. Change into the working tree and list the files present.

```
[terry@serverb ~]$ cd puppet
[terry@serverb puppet]$ ls -a
.  ..  .git
```

The only file found is the **.git** directory, because the changes made by **student** were committed into their local repository. The new files need to be pushed to the upstream repository.

11. Go back to **student**'s working tree on **servera**. Use the **git push** command to push the local repository changes to the upstream central repository. The changes must be published so others working on the project can use them.

```
[student@servera puppet]$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

No refs in common and none specified; doing nothing.
Perhaps you should specify a branch such as 'master'.
fatal: The remote end hung up unexpectedly
error: failed to push some refs to '/var/git/puppet.git'
 * [new branch]      master -> master
```

Git reports that it did not perform the push because there are a couple of ways that the push could be done. Define the **push.default** setting to **simple**, then try the push again.

```
[student@servera puppet]$ git config --global push.default simple
[student@servera puppet]$ git push
Counting objects: 6, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 2.41 KiB | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To /var/git/puppet.git
 * [new branch]      master -> master
```

12. Go back to **terry**'s working tree on **serverb**. Use the **git fetch** command to update the local repository with **student**'s content.

```
[terry@serverb puppet]$ git fetch
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From ssh://servera/var/git/puppet
 * [new branch]      master     -> origin/master
[terry@serverb puppet]$ ls -a
.  ..  .git
```

Although the local repository has the new content, **git fetch** does not change **terry**'s working tree.

13. The **git pull** command does the same thing as **git fetch**, except it updates the working tree. Use it now to update **terry**'s working tree.

```
[terry@serverb puppet]$ git pull
```

```
[terry@serverb puppet]$ ls
auth.conf  manifests  puppet.conf
```

14. As **terry**, make some changes to the working directory of the **puppet** project.

    14.1. Create a new file called **README** that has the following content:

    ```
    This is a file created by terry on serverb.
    ```

    14.2. Modify the value of the **mode** parameter in the **manifests/motd.pp** Puppet manifest to **644**. It should look like the following:

    ```
    file { '/etc/motd':
      ensure  => 'file',
      owner   => 'root',
      group   => 'root',
      mode    => '644',
      content => "No trespassing.\n",
    }
    ```

15. Use **git status** to display the status of the working tree. Notice how it differentiates between the new file and the modified file that is already under its control.

    ```
    [terry@serverb puppet]$ git status
    # On branch master
    # Changes not staged for commit:
    #   (use "git add <file>..." to update what will be committed)
    #   (use "git checkout -- <file>..." to discard changes in working directory)
    #
    #       modified:   manifests/motd.pp
    #
    # Untracked files:
    #   (use "git add <file>..." to include in what will be committed)
    #
    #       README
    no changes added to commit (use "git add" and/or "git commit -a")
    ```

16. The **git diff** command displays changes between modified files in the working tree and the most recent version checked into the repository.

    ```
    [terry@serverb puppet]$ git diff
    diff --git a/manifests/motd.pp b/manifests/motd.pp
    index f7cac14..0a55e77 100644
    --- a/manifests/motd.pp
    +++ b/manifests/motd.pp
    @@ -2,6 +2,6 @@ file { '/etc/motd':
       ensure  => 'file',
       owner   => 'root',
       group   => 'root',
    -  mode    => '444',
    +  mode    => '644',
       content => "No trespassing.\n",
     }
    ```

    README is not compared because it is not under Git's control.

17. The **`git log`** command displays the list of log messages used when committing changes to a file.

```
[terry@serverb puppet]$ git log README
[terry@serverb puppet]$ git log manifests/motd.pp
commit 55c6f7161443102dca9cd4bf04d651c8439f7361
Author: Student User <student@servera.lab.example.com>
Date:   Sat Oct 10 20:42:05 2015 -0400

    Initial /etc/puppet content
```

Again, **README** has nothing to display since it is not under Git's control.

18. Using the **`git show`** command with the commit ID of a file will display the changes made to that file during that commit. All the content of **`motd.pp`** was added during that initial commit.

```
[terry@serverb puppet]$ git show 55c6f7161443102dca9cd4bf04d651c8439f7361 manifests/
motd.pp
commit 55c6f7161443102dca9cd4bf04d651c8439f7361
Author: Student User <student@servera.lab.example.com>
Date:   Sat Oct 10 20:42:05 2015 -0400

    Initial /etc/puppet content

diff --git a/manifests/motd.pp b/manifests/motd.pp
new file mode 100644
index 0000000..f7cac14
--- /dev/null
+++ b/manifests/motd.pp
@@ -0,0 +1,7 @@
+file { '/etc/motd':
+  ensure  => 'file',
+  owner   => 'root',
+  group   => 'root',
+  mode    => '444',
+  content => "No trespassing.\n",
+}
```

19. Stage **README** for the next commit. Use the **`git status`** command to see how its state has changed.

```
[terry@serverb puppet]$ git add README
[terry@serverb puppet]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   manifests/motd.pp
#
```

The changes to **motd.pp** will not be saved to the repository with the next commit. It will be committed later.

20. Commit **README** to the repository.

```
[terry@serverb puppet]$ git commit
```

Since the log message was not specified on the command line, Git will open a text editor for entry of a log message. Add the following content, then save and exit the editor.

```
First edition of README.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   manifests/motd.pp
#
```

With the log information provided, Git will perform the commit.

```
[master 7a78d05] First edition of README.
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

21. Check the status of the working tree. Git states that **motd.pp** is not staged for a commit, and it offers a suggestion for committing it.

```
[terry@serverb puppet]$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   manifests/motd.pp
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Use the **git commit -a** to add and commit the changes in a single step. This time, specify a useful log message on the command line.

```
[terry@serverb puppet]$ git commit -a -m 'Changed file mode.'
[master c180199] Changed file mode.
 1 file changed, 1 insertion(+), 1 deletion(-)
```

22. Since these changes are complete and will keep the project moving forward, push them to the upstream repository.

    22.1. Set **terry**'s **push.default** setting to **simple**.

```
[terry@serverb puppet]$ git config --global push.default simple
```

    22.2. Push the changes to the central repository.

```
[terry@serverb puppet]$ git push
Counting objects: 10, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 706 bytes | 0 bytes/s, done.
Total 7 (delta 2), reused 0 (delta 0)
To ssh://servera/var/git/puppet.git
   55c6f71..c180199  master -> master
```

23. Go back to **student**'s working tree on **servera**. Pull the changes that Terry made into the local repository and the working tree.

```
[student@servera puppet]$ git pull
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (7/7), done.
From /var/git/puppet
   55c6f71..c180199  master     -> origin/master
Updating 55c6f71..c180199
Fast-forward
 README            | 1 +
 manifests/motd.pp | 2 +-
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 README
```

    Look carefully at the output. Git reports which files were changed in the update.

24. Now it is time to learn how to discard changes. Modify the **README** file and display the differences with the repository.

```
[student@servera puppet]$ echo more stuff >> README
[student@servera puppet]$ git diff
diff --git a/README b/README
index a15ca8d..6651bf9 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 This file was created by terry on serverb.
+more stuff
```

25. Use the following command sequence to see how to undo a change that has not been staged. **git reset** has no effect on the file's contents. The **git checkout --** command reverts the file to its original state.

```
[student@servera puppet]$ git reset README
Unstaged changes after reset:
```

```
M       README
[student@servera puppet]$ cat README
This file was created by terry on serverb.
more stuff
[student@servera puppet]$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
[student@servera puppet]$ git checkout -- README
[student@servera puppet]$ git status
# On branch master
nothing to commit, working directory clean
[student@servera puppet]$ cat README
This file was created by terry on serverb.
```

26. Use the following command sequence to see how to undo a change that has been staged. **`git reset`** unstages the file. It is the opposite of **`git add`**.

```
[student@servera puppet]$ echo 'another modification by student' >> README
[student@servera puppet]$ git add README
[student@servera puppet]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#
[student@servera puppet]$ git reset README
Unstaged changes after reset:
M       README
[student@servera puppet]$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Quiz: Managing Code with Git

Match the following Git commands to their descriptions in the table.

`git add`	`git clone`	`git commit`	`git config`
`git log`	`git pull`	`git push`	`git show`
`git status`			

Description	Command
Display the changes made to a file during a single commit.	
Display the current state of the working tree.	
Save and log staged changes to the local repository.	
Control the persistent settings for a Git user.	
Retrieve other developers' changes from the upstream repository into the local repository and the working tree.	
Display a list of messages about the changes made to a file.	
Check out the original content from an upstream Git repository.	

Description	Command
Stage a new or modified file for the next commit to the repository.	
Save changes to the upstream repository.	

Solution

Match the following Git commands to their descriptions in the table.

Description	Command
Display the changes made to a file during a single commit.	**`git show`**
Display the current state of the working tree.	**`git status`**
Save and log staged changes to the local repository.	**`git commit`**
Control the persistent settings for a Git user.	**`git config`**
Retrieve other developers' changes from the upstream repository into the local repository and the working tree.	**`git pull`**
Display a list of messages about the changes made to a file.	**`git log`**
Check out the original content from an upstream Git repository.	**`git clone`**
Stage a new or modified file for the next commit to the repository.	**`git add`**
Save changes to the upstream repository.	**`git push`**

Lab: Implementing Git

In this lab, you will create a shared Git repository and create a simple project to manage changes to files.

Resources	
Files	**/var/git** and **labwork/README**
Application URL	**http://materials.example.com/manifests/git-repo.pp**
Machines	**workstation**

Outcome(s)
You should be able to configure a shared Git repository and use it to manage a project.

Before you begin
You should have a Red Hat Enterprise Linux server installed with **yum** configured.

1. Log into **workstation** and run the lab setup script. It will configure Yum to use a repository that provides Puppet software.

```
[student@workstation ~]$ lab puppet-git setup
```

2. Create a system group, called **git**, that will determine who can manage projects in the shared repository. Make the **student** user a member of the **git** group. Create a shared Git repository on **workstation** under **/var/git**. Members of the **git** group should be able to create and administer projects in the repository.

Note

The Puppet manifest provided at **http://materials.example.com/manifests/git-repo.pp** will bring the system to the desired state.

3. Create a shared project in the repository called **labwork**.

4. Configure the default Git user name and email address for **student**. The user name should be "Student User" and the email address should be "student@workstation.lab.example.com".

5. Create a working directory in **~student** that is a clone of the **labwork** project.

6. Create a file in the **labwork** directory called **README** that contains the following text:

```
This is the README file.
```

Save the file so it is available in the shared repository for other users who may want to clone and work on the project.

7. Change the contents of the **README** file to include the following text:

```
This is the README file for the labwork project.
Important documentation will go here.
```

Save the file so it is available in the shared repository for other users who may want to clone and work on the project.

8. Run the lab grading script from **workstation** to check your work.

```
[student@workstation ~]$ lab puppet-git grade
```

9. On **workstation**, add the manifests from the previous labs into the **labwork** working tree. Commit and push them into the central Git repository on **workstation**.

10. As **root** on **servera**, clone the **ssh://student@workstation/var/git/labwork.git** repository. Add the manifests from the previous labs into the **labwork** working tree. Commit and push them into the central Git repository on **workstation**.

Solution

In this lab, you will create a shared Git repository and create a simple project to manage changes to files.

Resources	
Files	**/var/git** and **labwork/README**
Application URL	**http://materials.example.com/manifests/git-repo.pp**
Machines	**workstation**

Outcome(s)
You should be able to configure a shared Git repository and use it to manage a project.

Before you begin
You should have a Red Hat Enterprise Linux server installed with **yum** configured.

1. Log into **workstation** and run the lab setup script. It will configure Yum to use a repository that provides Puppet software.

```
[student@workstation ~]$ lab puppet-git setup
```

2. Create a system group, called **git**, that will determine who can manage projects in the shared repository. Make the **student** user a member of the **git** group. Create a shared Git repository on **workstation** under **/var/git**. Members of the **git** group should be able to create and administer projects in the repository.

Note

The Puppet manifest provided at **http://materials.example.com/manifests/git-repo.pp** will bring the system to the desired state.

Log into **workstation** as **root**, with the password **redhat**, to perform these initial system administrator tasks. Install the *puppet* package so the provided Puppet manifest can be used.

```
[root@workstation ~]# yum -y install puppet
... Output omitted ...
[root@workstation ~]# curl -O http://materials.example.com/manifests/git-repo.pp
... Output omitted ...
[root@workstation ~]# puppet apply git-repo.pp
Notice: Compiled catalog for workstation.lab.example.com in environment
 production in 0.47 seconds
Warning: The package type's allow_virtual parameter will be changing its
 default value from false to true in a future release. If you do not want
 to allow virtual packages, please explicitly set allow_virtual to false.
   (at /usr/share/ruby/vendor_ruby/puppet/type.rb:816:in `set_default')
Notice: /Stage[main]/Main/Package[git]/ensure: created
Notice: /Stage[main]/Main/Group[git]/ensure: created
Notice: /Stage[main]/Main/File[/var/git]/ensure: created
Notice: /Stage[main]/Main/User[student]/groups: groups changed 'wheel'
 to 'git,wheel'
Notice: Finished catalog run in 6.07 seconds
```

3. Create a shared project in the repository called **labwork**.

   Log out as **student** and log back in to inherit the **git** group privilege. Use **git init** to create the project.

```
[student@workstation ~]$ cd /var/git
[student@workstation git]$ git init --bare --shared=true labwork.git
Initialized empty shared Git repository in /var/git/labwork.git/
[student@workstation git]$ cd
```

4. Configure the default Git user name and email address for **student**. The user name should be "Student User" and the email address should be "student@workstation.lab.example.com".

```
[student@workstation ~]$ git config --global user.name 'Student User'
[student@workstation ~]$ git config --global user.email
 student@workstation.lab.example.com
```

5. Create a working directory in **~student** that is a clone of the **labwork** project.

```
[student@workstation ~]$ git clone /var/git/labwork.git
Cloning into 'labwork'...
warning: You appear to have cloned an empty repository.
done.
```

6. Create a file in the **labwork** directory called **README** that contains the following text:

```
This is the README file.
```

   Save the file so it is available in the shared repository for other users who may want to clone and work on the project.

   6.1. Change into the Git working directory, then create the **README** file with the specified content.

```
[student@workstation ~]$ cd labwork
[student@workstation labwork]$ vim README
[student@workstation labwork]$ cat README
This is the README file.
```

   6.2. Use **git add** to stage the file, then use **git commit** to save the file into the local repository with a log message.

```
[student@workstation labwork]$ git add README
[student@workstation labwork]$ git commit -m 'initial commit of README'
[master (root-commit) edd4965] initial commit of README
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

   6.3. Push the changes to the original, shared repository with the **git push** command. Be sure to define the **push.default** setting to **simple** before attempting the push.

```
[student@workstation labwork]$ git config --global push.default simple
```

```
[student@workstation labwork]$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 257 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /var/git/labwork.git
 * [new branch]      master -> master
[student@workstation labwork]$
```

7. Change the contents of the **README** file to include the following text:

```
This is the README file for the labwork project.
Important documentation will go here.
```

Save the file so it is available in the shared repository for other users who may want to clone and work on the project.

7.1. Make the specified changes to the file.

```
[student@workstation labwork]$ vi README
[student@workstation labwork]$ cat README
This is the README file for the labwork project.
Important documentation will go here.
```

7.2. Use **git add** and **git commit** to save the changes in the user's Git repository.

```
[student@workstation labwork]$ git add README
[student@workstation labwork]$ git commit -m 'expanded README content'
[master 8f59a57] expanded README content
 1 file changed, 2 insertions(+), 1 deletion(-)
```

7.3. Push the changes to the shared repository to make it available to other developers.

```
[student@workstation labwork]$ git push
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 347 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /var/git/labwork.git
   edd4965..8f59a57  master -> master
```

8. Run the lab grading script from **workstation** to check your work.

```
[student@workstation ~]$ lab puppet-git grade
```

9. On **workstation**, add the manifests from the previous labs into the **labwork** working tree. Commit and push them into the central Git repository on **workstation**.

9.1. Log into **workstation** as **student**. Define the **push.default** setting as **simple** for this user.

```
[student@workstation labwork]$ git config --global push.default simple
```

9.2. Move the **practice.pp** manifest into the **labwork** directory. Put it under control of Git and commit it with a meaningful log message.

```
[student@workstation labwork]$ mv ../practice.pp .
[student@workstation labwork]$ git add practice.pp
[student@workstation labwork]$ git commit -m 'Added practice.pp to labwork
 repo.'
[master 83bfb3f] Added practice.pp to labwork repo.
 1 file changed, 28 insertions(+)
 create mode 100644 practice.pp
```

9.3. Push the changes to the central repository.

```
[student@workstation labwork]$ git push
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 546 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /var/git/labwork.git
   1d18a9d..83bfb3f  master -> master
```

10. As **root** on **servera**, clone the **ssh://student@workstation/var/git/labwork.git** repository. Add the manifests from the previous labs into the **labwork** working tree. Commit and push them into the central Git repository on **workstation**.

10.1. Generate an SSH key pair and copy the public key to the **student@workstation** account.

```
[root@servera ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): Enter
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
59:c7:b3:be:af:0d:17:e2:ec:30:f3:1c:00:e8:d8:72 root@servera.lab.example.com
... Output omitted ...
[root@servera ~]# ssh-copy-id student@workstation.lab.example.com
The authenticity of host 'workstation.lab.example.com (172.25.250.254)'
 can't be established.
ECDSA key fingerprint is 47:49:14:fe:3c:31:f0:72:e7:ef:ed:6f:fd:4b:80:23.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
 filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
 prompted now it is to install the new keys
student@workstation.lab.example.com's password: student
... Output omitted ...
```

10.2. Define the global Git configurations for **root** on **servera**

```
[root@servera ~]# git config --global user.name 'Root User'
[root@servera ~]# git config --global user.email root@servera.lab.example.com
```

```
[root@servera ~]# git config --global push.default simple
```

10.3. Clone the **ssh://student@workstation/var/git/labwork.git** Git repository into **root**'s home directory.

```
[root@servera ~]# git clone ssh://student@workstation/var/git/labwork.git
Cloning into 'labwork'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 1), reused 0 (delta 0)
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.
```

10.4. Put the **ftp-setup.pp** and **test.pp** manifests under Git control. Move them into the **labwork** working tree.

```
[root@servera ~]# mv ftp-setup.pp test.pp labwork/
```

10.5. Change into the **labwork** working tree and check the current Git status of the files in it.

```
[root@servera ~]# cd labwork/
[root@servera labwork]# git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       ftp-setup.pp
#       test.pp
nothing added to commit but untracked files present (use "git add" to track)
```

10.6. Add the manifests to the staging area and commit them to the local Git repository with a meaningful log message.

```
[root@servera labwork]# git add *.pp
[root@servera labwork]# git commit -m 'Added manifests from servera.'
[master d401e79] Added manifests from servera.
 2 files changed, 59 insertions(+)
 create mode 100644 ftp-setup.pp
 create mode 100644 test.pp
```

10.7. Push the contents of the local Git repository to the central repository.

```
[root@servera labwork]# git push
Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 788 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To ssh://student@workstation/var/git/labwork.git
   7101b88..d401e79  master -> master
```

10.8. Confirm the files from **servera** are in the repository. As **student**, go back to the **labwork** directory and pull the updates from the central Git repository.

```
[student@workstation labwork]$ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From /var/git/labwork
   7101b88..d401e79  master     -> origin/master
Updating 7101b88..d401e79
Fast-forward
 ftp-setup.pp | 31 ++++++++++++++++++++++++++++++
 test.pp      | 28 ++++++++++++++++++++++++++++
 2 files changed, 59 insertions(+)
 create mode 100644 ftp-setup.pp
 create mode 100644 test.pp
[student@workstation labwork]$ ls
ftp-setup.pp  practice.pp  README  test.pp
```

Summary

In this chapter, you learned:

- Implementing a shared Git repository begins with the creation of a dedicated group for Git users, and possibly a **`git`** user. The repository must be a public directory owned by that group with the set group ID permission set.
- The **`git init`** command creates new projects in a Git repository.
- Before working on projects, a user must define their name and email address for Git commits with **`git config`**.
- A user begins working on a project by cloning the project from a shared Git repository with the **`git clone`** command.
- When files are created or modified, they are added to the staging area with the **`git add`** command.
- The **`git commit`** command saves staged files into the local repository with a log message.
- Changes made by other users are fetched from the shared Git repository with the **`git pull`** command.
- Changes made in the local Git repository are saved to the original shared repository with the **`git push`** command.

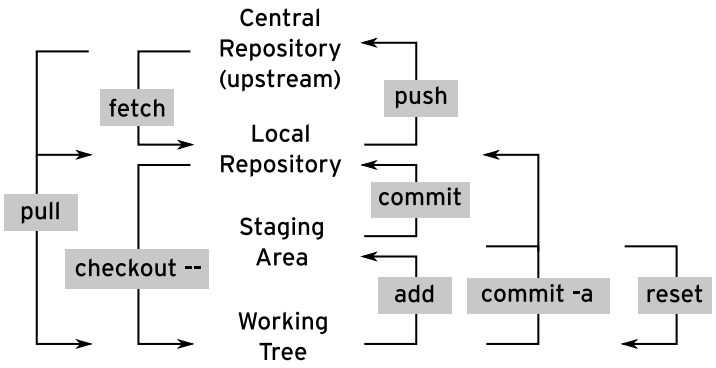

Figure 5.6: Summary of commonly used Git subcommands

CHAPTER 6

DISPLAYING SYSTEM INFORMATION WITH FACTER

Overview	
Goal	Display information about systems using Facter.
Objectives	• Display system facts with Facter. • Add new facts to Facter.
Sections	• Displaying System Facts with Facter (and Guided Exercise) • Adding New Facts to Facter (and Guided Exercise)
Lab	• Displaying System Information with Facter

Displaying System Facts with Facter

Objectives

After completing this section, students should be able to:

- Display system facts with **`facter`**.

Identifying system facts

Puppet uses system facts about client systems to determine how to configure them. The first thing the Puppet agent does when it begins a Puppet run is identify system facts about the node it is running on. It sends those facts to the Puppet master when the agent contacts it. The Puppet master uses those facts to compile the catalog that determines the manifests that will be applied on the Puppet client.

Some of the facts that Puppet gathers about a node includes the operating system it runs, hardware information, and network information, such as the host name and IP address. These facts can be used by Puppet to generate host-specific information in configuration files. They are also used by Puppet to make decisions.

The **`facter`** command can be used at the shell prompt to display system facts. Without arguments, **`facter`** prints a list of all the key/value pairs that describe a system.

```
[root@host ~]# facter | head
architecture => x86_64
augeasversion => 1.0.0
bios_release_date => 01/01/2011
bios_vendor => Bochs
bios_version => Bochs
blockdevice_vda_size => 107374182400
blockdevice_vda_vendor => 6900
blockdevices => vda
domain => example.com
facterversion => 1.7.6
```

When given a single argument, **`facter`** displays the value of the corresponding system fact. Nothing is displayed when a fact is specified that does not exist.

```
[root@host ~]# facter memorysize
3.74 GB
[root@host ~]# facter cpuspeed
[root@host ~]#
```

Using system facts in Puppet DSL

System facts are often referenced in Puppet DSL. The following Puppet DSL snippet has a condition that depends on a system fact:

```
[root@host ~]# cat facter-example.pp
if $::osfamily == 'redhat' {
  $fpath = '/tmp/example'
}
else {
  $fpath = '/var/tmp/example'
```

```
}
file { 'example':
  path    => $fpath,
  ensure  => file,
  mode    => 0444,
  owner   => 'root',
  group   => 'root',
  content => "\nIMPORTANT DATA.\n\n",
}
```

$::osfamily is a reference to a system fact. The **facter osfamily** command displays its value.

```
[root@host ~]# facter osfamily
RedHat
```

Although the value has mixed case, the comparison in the Puppet DSL is case insensitive. Consider what happens when the **facter-example.pp** manifest is applied.

```
[root@host ~]# puppet apply facter-example.pp
Notice: Compiled catalog for host.example.com in environment production
 in 0.23 seconds
Notice: /Stage[main]/Main/File[example]/ensure: defined content as
 '{md5}1d8b8c9d7e971f61687a3a552afc0cae'
Notice: Finished catalog run in 0.06 seconds
[root@host ~]# ls /tmp/example
/tmp/example
[root@host ~]# ls /var/tmp/example
ls: cannot access /var/tmp/example: No such file or directory
[root@host ~]# cat /tmp/example

IMPORTANT DATA.
```

The first clause of the conditional statement matched the value of the **$::osfamily** with 'redhat', so the file name was set to **/tmp/example**. That is the file that was created when the manifest was applied.

Useful system facts

The following are some of the useful system facts identified by **facter**.

Hardware architecture information can be obtained.

```
[root@host ~]# facter architecture
x86_64
```

Information can be identified about the user running the **facter** command.

```
[root@host ~]# facter id
root
```

Network information can be obtained. This includes the fully qualified domain name of the host. Information about the network interfaces and their corresponding IP addresses can also be identified.

```
[root@host ~]# facter fqdn
```

```
host.example.com
[root@host ~]# facter interfaces
interfaces => eth0,lo
[root@host ~]# facter | grep ipaddress
ipaddress => 172.25.250.10
ipaddress_eth0 => 172.25.250.10
ipaddress_lo => 127.0.0.1
```

Information about the operating system release can be identified.

```
[root@host ~]# facter | grep operating
operatingsystem => RedHat
operatingsystemmajrelease => 7
operatingsystemrelease => 7.1
[root@host ~]# facter osfamily
RedHat
```

facter can determine if it is running on a virtual host, and if so, what type of virtualization is being used.

```
[root@host ~]# facter | grep virtual
is_virtual => true
virtual => kvm
```

References

facter(8) man page

Guided Exercise: Displaying System Facts with Facter

In this lab, you will use the **facter** command to display facts about a Red Hat Enterprise Linux system.

Resources	
Machines	**servera**

Outcome(s)
You should be able to use the **facter** command to display facts about a host.

Before you begin
The *facter* package should already be installed on your **servera** host.

1. Log in as **root** on **servera** and confirm the package that provides the **facter** command is installed on the system.

```
[root@servera ~]# yum -y install facter
```

2. Use the **facter** command to display the list of informative values for the system.

```
[root@servera ~]# facter
architecture => x86_64
augeasversion => 1.1.0
bios_release_date => 01/01/2011
bios_vendor => Bochs
bios_version => Bochs
blockdevice_fd0_size => 0
blockdevice_sr0_model => QEMU DVD-ROM
blockdevice_sr0_size => 1073741312
blockdevice_sr0_vendor => QEMU
blockdevice_vda_size => 10737418240
... Output omitted ...
```

3. What **facter** commands will display the host name of the system?

```
[root@servera ~]# facter hostname
servera
[root@servera ~]# facter fqdn
servera.lab.example.com
```

4. What **facter** command would you use to display the IP address of the system?

```
[root@servera ~]# facter ipaddress
172.25.250.10
```

5. The **facter** command displays different information when it is used by different users.

   5.1. Execute **facter** without arguments as **root** and save the output to a file.

```
[root@servera ~]# facter > /tmp/root.facts
```

5.2. Execute **facter** without arguments as **student** and save the output to a file.

```
[root@servera ~]# su - student
[student@servera ~]$ facter > /tmp/student.facts
```

5.3. Use the **diff** command to compare the system facts the two users can display.

```
[student@servera ~]$ diff /tmp/root.facts /tmp/student.facts
3,5d2
< bios_release_date => 01/01/2011
< bios_vendor => Bochs
< bios_version => Bochs
20c17
< id => root
---
> id => student
25c22
< is_virtual => true
---
> is_virtual => false
32d28
< manufacturer => Bochs
34c30
< memoryfree_mb => 1671.92
---
> memoryfree_mb => 1670.11
47c43
< path => /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
---
> path => /usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/
student/.local/bin:/home/student/bin
51d46
< productname => Bochs
63d57
< serialnumber => Not Specified
75d68
< type => Other
80,82c73,74
< uptime_seconds => 30624
< uuid => Not Settable
< virtual => kvm
---
> uptime_seconds => 30637
> virtual => physical
```

Some of the differences are due to system changes (**memoryfree_mb** and **uptime_seconds** for example), but other differences are explained because of differences in system-level access (**bios_*** and **is_virtual** facts).

Adding New Facts to Facter

Objectives

After completing this section, students should be able to:

- Add new facts to Facter.

Adding new facts in a Linux environment

Facter has the capability for providing additional custom facts. It can be as simple as creating structured files or scripts, or it can be as complex as writing custom Ruby code. This section will take a look at how to create additional system facts using structured files and scripts.

Facter looks for additional definitions for facts in the following two directories: **`/etc/facter/facts.d`** and **`/etc/puppetlabs/facter/facts.d/`**. The structured files and scripts that define new facts should be placed in these directories. These directories are not created by the *facter* package.

Structured files

Facter currently supports three types of structured data files: text, YAML, and JSON. The file name for these files is important because Facter uses the file suffix (**`.txt`**, **`.yaml`**, and **`.json`** respectively) to determine what type of structured data is contained in the file.

The most simple structured file is a text file. It must end with a **`.txt`** extension. Each line of the file contains a key/value pair in a format similar to the following:

```
key1=value1
key2=value2
key3=value3
...
keyN=valueN
```

The YAML structured file must end with a **`.yaml`** extension and must contain valid YAML syntax. String values are enclosed in single or double quotes. The following is an example YAML structured file:

```
---
key1: 'value1'
key2: 'value2'
key3: 'value3'
...
keyN: 'valueN'
```

The JSON structured file must end with a **`.json`** extension and must contain valid JSON syntax. String values are enclosed in double quotes. The following is an example JSON structured file:

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3",
  ...
  "keyN": "valueN"
```

```
}
```

Executable programs

Executable programs can provide Facter with dynamic facts that are computed or collected dynamically. They can be written in any programming language. The file in **facts.d** must be executable for Facter to execute and use it.

The facts generated by executable programs must be printed as name/value pairs in a format that is consistent with the text structured file format. JSON and YAML output is not recognized in this context.

The following example uses a Bash shell script to obtain system facts. This example uses the **who** and **wc** commands to calculate the number of users logged in. A system fact, named **custom_num_users** is defined.

```
[root@host ~]# cat /etc/facter/facts.d/custom
#!/bin/bash
PATH=/bin:/usr/bin
echo "custom_num_users=$(who | wc -l)"
[root@host ~]# chmod a+x /etc/facter/facts.d/custom
[root@host ~]# /etc/facter/facts.d/custom
custom_num_users=1
[root@host ~]# facter custom_num_users
1
```

The following is a C program that displays a static name/value pair. It is compiled and the resulting executable is placed in the **/etc/facter/facts.d** directory. Although the example program prints a fact that does not change, it could have calculated a run time value and returned that instead.

```
[root@host ~]# cat custom-hello.c
#include <stdio.h>

void main (void)
{
    printf("custom_hello=Hello world!\n");
}
[root@host ~]# gcc -o /etc/facter/facts.d/custom-hello custom-hello.c
[root@host ~]# facter custom_hello
Hello world!
```

Troubleshooting

If external facts that have been defined are not being displayed, Facter has a debug mode that will print informative messages that can help diagnose the problem. Invoke **facter** with the **--debug** option to display these messages.

The following example shows what happens when a structured file does not have a proper key/value pair definition:

```
[root@host ~]# facter --debug test
Fact file /etc/facter/facts.d/test.txt was parsed but returned an empty
data set
value for lsbdistid is still nil
Not an EC2 host
[root@host ~]# cat /etc/facter/facts.d/test.txt
```

```
test-This is a test
```

The **--debug** option to **facter** can also help identify executable programs that generate invalid key/value pairs in their output:

```
[root@host ~]# facter --debug test
Fact file /etc/facter/facts.d/test was parsed but returned an empty
data set
value for lsbdistid is still nil
Not an EC2 host
[root@host ~]# ls -l /etc/facter/facts.d/test
-rwxr-xr-x. 1 root root 37 Aug 24 09:53 /etc/facter/facts.d/test
[root@host ~]# cat /etc/facter/facts.d/test
#!/bin/bash
echo test-This is a test
```

References

For more information, see

> Puppet Labs - Facter 3.0: Custom Facts Walkthrough
> http://docs.puppetlabs.com/guides/custom_facts.html

Guided Exercise: Adding New Facts to Facter

In this practice exercise, you will configure your system so that it publishes custom facts about your system for use by the **facter** command.

Resources	
Machines	**servera**

Outcome(s)
You should be able to configure a system to provide custom variable/value pairs for use by **facter**.

Before you begin
The *facter* package should already be installed on your **servera** host.

1. Log in as **root** on **servera**. Make the directory that **facter** looks for custom facts in, **/etc/facter/facts.d**.

```
[root@servera ~]# mkdir -p /etc/facter/facts.d
```

2. Custom facts can be added with the use of files that end with a **.txt** extension.

   2.1. Use an editor to create a file, **custom.txt**, with two new custom facts. It should contain the following content.

```
custom_fact1=value one
custom_fact2=value two
```

   2.2. Confirm the new facts are available to **facter**.

```
[root@servera ~]# facter | grep custom
custom_fact1 => value one
custom_fact2 => value two
```

3. Custom facts can be added with the use of files that end with a **.yaml** extension.

   3.1. Use an editor to create a file, **custom.yaml**, with two new custom facts. The file should contain the following content.

```
---
custom_fact3: 'value three'
custom_fact4: 'value four'
```

   3.2. Confirm the new facts are available to **facter**.

```
[root@servera ~]# facter | grep custom
... Output omitted ...
custom_fact3 => value three
custom_fact4 => value four
```

4. Custom facts can be added with the use of files that end with a **.json** extension.

   4.1. Use an editor to create a file, **custom.json**, with two new custom facts. The file should contain the following.

```
{"custom_fact5": "value five", "custom_fact6": "value six"}
```

   4.2. Confirm the new facts are available to **facter**.

```
[root@servera ~]# facter | grep custom
... Output omitted ...
custom_fact5 => value five
custom_fact6 => value six
```

5. Custom facts can be added with the use of shell programs. These are files found in the **/etc/facter/facts.d** directory that are executable.

   5.1. Use an editor to create a file, **custom**, with a new custom fact. The fact should be called **custom_realtime_seconds** that has the current system time in seconds. The following shell script will produce the desired output.

```
#!/bin/bash
echo "custom_realtime_seconds=$(date '+%s')"
```

   5.2. Make the shell script executable.

```
[root@servera ~]# chmod 755 /etc/facter/facts.d/custom
```

   5.3. Confirm the new fact is available to **facter**.

```
[root@servera ~]# facter | grep custom
... Output omitted ...
custom_realtime_seconds => 1439819414
```

   5.4. Wait a few seconds and display the value of **custom_realtime_seconds**. It should be calculated each time the fact is checked.

```
[root@servera ~]# facter | grep realtime
custom_realtime_seconds => 1439819466
```

Lab: Displaying System Information with Facter

In this lab, you will configure your system so that it publishes custom facts about your system for use by the **facter** command.

Resources	
Files	**/etc/rht**
Machines	**servera**

Outcome(s)
You should be able to configure a system to provide custom variable/value pairs for use by **facter**.

Before you begin
The *facter* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-facter setup
```

2. Log in as **root** on **servera** and configure **facter** to publish the values for the shell variables defined in **/etc/rht**. The system facts should have the same name and the same values as the original variables.

3. Use the **facter** command to check your work.

4. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-facter grade
```

Solution

In this lab, you will configure your system so that it publishes custom facts about your system for use by the **facter** command.

Resources	
Files	**/etc/rht**
Machines	**servera**

Outcome(s)
You should be able to configure a system to provide custom variable/value pairs for use by **facter**.

Before you begin
The *facter* package should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script.

```
[student@workstation ~]$ lab puppet-facter setup
```

2. Log in as **root** on **servera** and configure **facter** to publish the values for the shell variables defined in **/etc/rht**. The system facts should have the same name and the same values as the original variables.

   The following solution makes use of a text file. YAML, JSON, or a shell script can also be used.

```
[root@servera ~]# mkdir -p /etc/facter/facts.d
[root@servera ~]# vi /etc/facter/facts.d/rht.txt
[root@servera ~]# cat /etc/facter/facts.d/rht.txt
rht_venue=venue
rht_role=servera
rht_vmtree=puppet3.6/x86_64
rht_network=yes
... Output omitted ...
```

3. Use the **facter** command to check your work.

```
[root@servera ~]# facter rht_vmtree
puppet3.6/x86_64
```

4. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-facter grade
```

Summary

In this chapter, you learned:

- The **facter** command displays a limited set of facts about a host system.
- Additional system key/value pairs can be defined in structured text, YAML, and JSON files in **/etc/facter/facts.d**.
- Executable programs that generate key/value pairs can also be placed in **/etc/facter/facts.d**.

CHAPTER 7

IMPLEMENTING PUPPET MODULES

Overview	
Goal	Create Puppet modules and implement classes in a manifest.
Objectives	• Create a Puppet module structure. • Design a Puppet manifest with a class. • Deploy a Puppet module using a smoke test.
Sections	• Building Puppet Modules (and Guided Exercise) • Implementing Classes (and Guided Exercise) • Deploying a Puppet Module Using a Smoke Test (and Guided Exercise)
Lab	• Implementing Puppet Modules

Building Puppet Modules

Objectives

After completing this section, students should be able to:

- Create a Puppet module structure.

Writing Puppet modules

Creating Puppet modules is a way of packaging Puppet manifests and related files into a single file. The module is a tar archive that has an established directory hierarchy. The **puppet module** command helps develop, build, install, and manage Puppet modules on a system.

Exploring a module directory structure

Puppet modules provide a standard, predictable directory structure for the Puppet code and related files that they contain. When a module is being developed, the top-level directory of a Puppet module combines the name of the module author with the module name. After a module has been built and distributed, the top-level directory will also include the version number of the module.

Some of the essential or most frequently used directories in a module hierarchy are listed below:

- The **manifests** directory contains all of the Puppet manifests defined in the module. It always contains an **init.pp** manifest, which is the "main" manifest of the module.
- The **files** directory contains static files. Often these are configuration files used by users or services created by the module manifests.
- The **lib/facter** directory contains custom fact definitions. These should be facts that are used by the manifests in the module.
- The **tests** directory contains the manifests that are used to test the functionality provided by the module. It also contains an **init.pp** manifest, which tests the "main" manifest of the module.

Exploring the metadata.json file

Every module includes a **metadata.json** information file in its top-level directory. This file is a JSON-formatted file that defines name/value pairs of metadata that describes the module's version, its purpose, and points to additional URLs where more documentation and contact information are provided.

The following is a minimal **metadata.json** file:

```
[user@host ~]$ cat rht-modname/metadata.json
{
  "name": "rht-modname",
  "version": "0.1.0",
  "author": "rht",
  "summary": null,
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
```

```
  "dependencies": [
    {
      "name": "puppetlabs-stdlib",
      "version_range": ">= 1.0.0"
    }
  ]
}
```

The example also demonstrates how a module dependency is defined. This particular module is dependent on the function library provided by the **puppetlabs-stdlib** module.

Using `puppet module generate`

The **puppet module generate** command creates a skeleton of a working directory for writing a module. The syntax for this command is:

```
[user@host ~]$ puppet module generate author-modulename
```

The *author* of the module name specifies the author of the module and the *modulename* specifies the name of the module being written. This command creates a directory structure for writing the module below the current directory, named *author-modulename*.

```
[user@host ~]$ puppet module generate rht-modname
We need to create a metadata.json file for this module.  Please answer
the following questions; if the question is not applicable to this module,
feel free to leave it blank.

Puppet uses Semantic Versioning (semver.org) to version modules.
What version is this module?  [0.1.0]
--> Enter

Who wrote this module?  [rht]
--> Enter

... Output omitted ...
```

puppet module generate asks a few questions about the module (version, license, brief description, contact email, etc.). Default values are provided, so typing Enter at any of the prompts will accept the proposed default value. The questions can be skipped if **puppet module generate** is invoked with the **--skip-interview** option.

Building Puppet modules

The **puppet module build** command takes a Puppet module working directory and packages it into a tar archive. The top-level directory is passed as the argument to this command. The module is built and packaged in the **pkg** directory in the module's working directory.

```
[root@host ~]# puppet module build rht-modname
Notice: Building /root/rht-modname for release
Module built: /root/rht-modname/pkg/rht-modname-0.1.0.tar.gz
[root@host ~]# ls -F rht-modname/pkg
rht-modname-0.1.0/  rht-modname-0.1.0.tar.gz
```

The **puppet module install** command installs a Puppet module in the **/etc/puppet/modules** directory on the system. It must be executed by **root**. Puppet will create the **/etc/puppet/modules** directory if it does not already exist.

```
[root@host ~]# puppet module install rht-modname/pkg/rht-modname-0.1.0.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└─┬ rht-modname (v0.1.0)
  └── puppetlabs-stdlib (v4.9.0)
```

After the module is installed, **puppet module install** will also display any dependencies that the module has. The module in the example depends on **puppetlabs-stdlib**, which is already installed on the system.

References

puppet-module(8) man page

For more information, see

> Puppet Labs - Module Fundamentals
> http://docs.puppetlabs.com/puppet/3.6/reference/modules_fundamentals.html

Guided Exercise: Building Puppet Modules

In this exercise, you will write a simple Puppet module without any dependencies.

Resources	
Files	**labwork/test.pp**, **manifests/init.pp**, and **metadata.json**
Application URL	**http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**, **http://materials.example.com/manifests/test.pp** and **ssh://student@workstation/var/git/labwork.git**
Machines	**workstation** and **servera**

Outcome(s)
You should be able to package an existing Puppet manifest as a Puppet module.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host. You should also have a Puppet manifest from an earlier practice exercise that implements a web server, named **labwork/test.pp**. If **test.pp** does not exist on your system, download a copy from **http://materials.example.com/manifests/test.pp**.

1. Log into **workstation** as **student** and download a simple module from **http:///materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**.

```
[student@workstation ~]$ curl -O http://materials.example.com/modules/thoraxe-
motd-0.1.1.tar.gz
... Output omitted ...
[student@workstation ~]$ file thoraxe-motd-0.1.1.tar.gz
thoraxe-motd-0.1.1.tar.gz: gzip compressed data, from Unix, last modified:
 Fri Mar 21 16:18:31 2014
```

2. Extract the files from the archive and change into the top-level directory of the module.

```
[student@workstation ~]$ tar xvf thoraxe-motd-0.1.1.tar.gz
thoraxe-motd-0.1.1/
thoraxe-motd-0.1.1/Modulefile
thoraxe-motd-0.1.1/tests/
thoraxe-motd-0.1.1/tests/init.pp
thoraxe-motd-0.1.1/manifests/
thoraxe-motd-0.1.1/manifests/init.pp
thoraxe-motd-0.1.1/README
thoraxe-motd-0.1.1/metadata.json
thoraxe-motd-0.1.1/spec/
thoraxe-motd-0.1.1/spec/spec_helper.rb
[student@workstation ~]$ cd thoraxe-motd-0.1.1
```

3. Examine the **manifests/init.pp** file. It contains the main Puppet DSL of the Puppet module.

```
[student@workstation thoraxe-motd-0.1.1]$ cat manifests/init.pp
```

```
# == Class: motd
#
# Full description of class motd here.
#
... Output omitted ...
# === Copyright
#
# Copyright 2014 Your name here, unless otherwise noted.
#
class motd ($message = 'This is the default message') {

  file { '/etc/motd':
    content => "$message \n",
  }

}
```

4. Log into **servera** as **student** (password of **student**). To avoid repeatedly typing passwords, generate an SSH key pair and upload the public key to **student**'s account on **workstation**.

```
[student@servera ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_rsa): Enter
Enter passphrase (empty for no passphrase): Enter
Enter same passphrase again: Enter
Your identification has been saved in /home/student/.ssh/id_rsa.
Your public key has been saved in /home/student/.ssh/id_rsa.pub.
The key fingerprint is:
a4:fb:38:bb:95:f1:f8:4d:4d:67:0f:54:53:17:59:00 student@servera.lab.example.com
... Output omitted ...
[student@servera ~]$ ssh-copy-id workstation
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
 filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
 prompted now it is to install the new keys
student@workstation's password: student
... Output omitted ...
```

5. Define global Git configuration settings for **student**.

```
[student@servera ~]$ git config --global user.name 'Student User'
[student@servera ~]$ git config --global user.email student@servera.lab.example.com
[student@servera ~]$ git config --global push.default simple
```

6. Clone the **ssh://student@workstation/var/git/labwork.git** Git repository. Change into it because all development work will be done in that directory.

```
[student@servera ~]$ git clone ssh://workstation/var/git/labwork.git
Cloning into 'labwork'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 2), reused 0 (delta 0)
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (2/2), done.
[student@servera ~]$ cd labwork
```

7. Use the **puppet module generate** command to create a new working directory for the module you are going to create. Any questions that are answered with an empty response accept the default value presented by the tool. Provide a simple description of this module when prompted.

```
[student@servera labwork]$ puppet module generate rht-webtest
We need to create a metadata.json file for this module.  Please answer the
following questions; if the question is not applicable to this module, feel free
to leave it blank.

Puppet uses Semantic Versioning (semver.org) to version modules.
What version is this module?  [0.1.0]
--> Enter

Who wrote this module?  [rht]
--> Enter

What license does this module code fall under?  [Apache 2.0]
--> Enter

How would you describe this module in a single sentence?
--> Red Hat Training test web server.

Where is this module's source code repository?
--> Enter

Where can others go to learn more about this module?
--> Enter

Where can others go to file issues about this module?
--> Enter

----------------------------------------
{
  "name": "rht-webtest",
  "version": "0.1.0",
  "author": "rht",
  "summary": "Red Hat Training test web server.",
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
    {
      "name": "puppetlabs-stdlib",
      "version_range": ">= 1.0.0"
    }
  ]
}
----------------------------------------

About to generate this metadata; continue? [n/Y]
--> y

Notice: Generating module at /home/student/labwork/rht-webtest...
Notice: Populating ERB templates...
Finished; module generated in rht-webtest.
rht-webtest/Rakefile
rht-webtest/manifests
rht-webtest/manifests/init.pp
rht-webtest/spec
rht-webtest/spec/classes
```

```
rht-webtest/spec/classes/init_spec.rb
rht-webtest/spec/spec_helper.rb
rht-webtest/tests
rht-webtest/tests/init.pp
rht-webtest/README.md
rht-webtest/metadata.json
```

8. Put the new module directory, **rht-webtest**, under control of Git. Commit it into the local repository with a descriptive log message.

```
[student@servera labwork]$ git add rht-webtest
[student@servera labwork]$ git commit -m 'Initial version of rht-webtest.'
[master 90d6d34] Initial version of rht-webtest.
 7 files changed, 191 insertions(+)
 create mode 100644 rht-webtest/README.md
 create mode 100644 rht-webtest/Rakefile
 create mode 100644 rht-webtest/manifests/init.pp
 create mode 100644 rht-webtest/metadata.json
 create mode 100644 rht-webtest/spec/classes/init_spec.rb
 create mode 100644 rht-webtest/spec/spec_helper.rb
 create mode 100644 rht-webtest/tests/init.pp
```

9. Change into the working directory for the module that was created by **puppet module generate** and explore.

```
[student@servera labwork]$ cd rht-webtest/
[student@servera rht-webtest]$ ls -F
manifests/  metadata.json  Rakefile  README.md  spec/  tests/
```

Inspect the generated content of the main Puppet manifest, **manifests/init.pp**.

```
[student@servera rht-webtest]$ ls manifests
init.pp
[student@servera rht-webtest]$ cat manifests/init.pp
# == Class: webtest
#
# Full description of class webtest here.
#
# === Parameters
#
# Document parameters here.
#
# [*sample_parameter*]
#   Explanation of what this parameter affects and what it defaults to.
#   e.g. "Specify one or more upstream ntp servers as an array."
#
... Output omitted ...
# === Copyright
#
# Copyright 2015 Your name here, unless otherwise noted.
#
class webtest {


}
```

10. Make the resources of the existing **test.pp** Puppet manifest the main Puppet manifest for the module. Remove or comment the boilerplate Puppet DSL in **manifests/init.pp**

and add the web server Puppet resources instead. The **test.pp** manifest can be found in the **labwork** directory since it was checked into Git earlier. Remove the **notify** resource since it will no longer be used. If you do not have access to an existing manifest, then the necessary Puppet DSL is listed as follows.

```
# Copyright 2015 Your name here, unless otherwise noted.
#

user { 'webmaster':
  ensure  => 'present',
  home    => '/home/webmaster',
  shell   => '/bin/bash',
}

file { '/home/webmaster':
  ensure  => 'directory',
  owner   => 'webmaster',
  group   => 'webmaster',
  mode    => '0770',
  require => User['webmaster'],
}

package { 'httpd':
  ensure  => 'present',
}

service { 'httpd':
  ensure  => 'running',
  enable  => true,
  require => Package['httpd'],
}
```

Add the updated **init.pp** manifest to the Git staging area.

```
[student@servera rht-webtest]$ git add manifests/init.pp
```

11. Many module writers use library functions from the **puppetlabs-stdlib** Puppet module. The **puppet manifest generate** command assumes this is the case, so it creates a JSON list that includes this module in the **metadata.json** by default.

    Since the Puppet manifest that is used in our example is self-contained, remove the **puppetlabs-stdlib** reference and make the **dependencies** value an empty list. The following should be the contents of the resulting **metadata.json** file.

```
{
  "name": "rht-webtest",
  "version": "0.1.0",
  "author": "rht",
  "summary": "Red Hat Training test web server.",
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
  ]
}
```

Add the updated **metadata.json** file to the Git staging area.

```
[student@servera rht-webtest]$ git add metadata.json
```

12. Change to the parent directory of the module's working directory and use the **puppet module build** command to assemble the Puppet components in the working directory into a module.

```
[student@servera rht-webtest]$ cd ..
[student@servera labwork]$ puppet module build rht-webtest
Notice: Building /home/student/labwork/rht-webtest for release
Module built: /home/student/labwork/rht-webtest/pkg/rht-webtest-0.1.0.tar.gz
```

A new directory exists in the working directory named **pkg**. It contains a tar archive that is the resulting module from the build.

```
[student@servera labwork]$ ls -l rht-webtest/pkg
total 8
drwxrwxr-x. 5 student student 4096 Sep 15 19:36 rht-webtest-0.1.0
-rw-rw-r--. 1 student student 3631 Sep 15 19:36 rht-webtest-0.1.0.tar.gz
[student@servera labwork]$ tar tzf rht-webtest/pkg/rht-webtest-0.1.0.tar.gz
rht-webtest-0.1.0/
rht-webtest-0.1.0/Rakefile
rht-webtest-0.1.0/manifests/
rht-webtest-0.1.0/manifests/init.pp
rht-webtest-0.1.0/spec/
rht-webtest-0.1.0/spec/classes/
rht-webtest-0.1.0/spec/classes/init_spec.rb
rht-webtest-0.1.0/spec/spec_helper.rb
rht-webtest-0.1.0/tests/
rht-webtest-0.1.0/tests/init.pp
rht-webtest-0.1.0/README.md
rht-webtest-0.1.0/metadata.json
rht-webtest-0.1.0/checksums.json
```

13. Commit the changes to the local Git repository with a meaningful log message. Push them to the upstream repository once the **rht-webtest** Puppet module successfully builds.

```
[student@servera labwork]$ git commit -m 'First edition of rht-webtest module.'
[master 79fa396] First edition of rht-webtest module.
 2 files changed, 20 insertions(+), 5 deletions(-)
[student@servera labwork]$ git push
Counting objects: 21, done.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (20/20), 4.46 KiB | 0 bytes/s, done.
Total 20 (delta 4), reused 0 (delta 0)
To ssh://workstation/var/git/labwork.git
   d401e79..79fa396  master -> master
```

Implementing Classes

Objectives

After completing this section, students should be able to:

- Design a Puppet manifest with a class.

Reusing Puppet code with classes

Puppet manifests specify how a particular system is supposed to be configured, but there is a way that Puppet resource definitions can be made more general and reusable so that they can be applied to multiple hosts: Puppet classes.

A Puppet class defines a named block of Puppet resource definitions for later use. In a way, they are similar to function definitions in programming languages. A Puppet class usually defines all of the resources needed to implement a service or run an application (users, packages, config files, services). Puppet classes can be combined into higher-level classes that define a system role, such as FTP server.

The following example demonstrates the minimal syntax of a Puppet class definition:

```
class class_name {
  resource definitions...
}
```

The **`class`** keyword is followed by the name of the class. Optionally, class parameters would follow the class name, enclosed in parentheses. Braces enclose the code that defines the Puppet class. Typically this is a list of resource definitions, possibly defining all of the resources needed to provide a network service. The following shows how a class that takes parameters would be defined:

```
class class_name ($param = 'value') {
  resource definitions...
}
```

The **`include`** command is used in a manifest to use a class. It is similar to calling a previously defined function in a programming language.

In modules, Puppet classes are defined in the files found in the **`manifests`** directory, one class per file. Each file should have the same name as the class it defines. The **`init.pp`** file should define a class with the same name as the module name.

References

For more information, see

> Puppet Labs - Language: Classes
> http://docs.puppetlabs.com/puppet/3.6/reference/lang_classes.html

Guided Exercise: Implementing Classes

In this lab, you will make a module more reusable by having it publish a Puppet class instead of a manifest.

Resources	
Files	**manifests/init.pp**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet module that implements a reusable Puppet class.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host. You should also have a Puppet module from an earlier practice exercise that implements a web server, named **rht-webtest**.

1. Log in as **student** on **servera** (password is **student**). Edit the main manifest for the **rht-webtest** module. Modify **manifests/init.pp** to change the existing Puppet DSL into a Puppet class called **webtest**. The class definition should look similar to the following.

```
#
# Copyright 2015 Your name here, unless otherwise noted.
#

class webtest {

  user { 'webmaster':
    ensure  => 'present',
    home    => '/home/webmaster',
    shell   => '/bin/bash',
  }

  file { '/home/webmaster':
    ensure  => 'directory',
    owner   => 'webmaster',
    group   => 'webmaster',
    mode    => '0770',
    require => User['webmaster'],
  }

  package { 'httpd':
    ensure  => 'present',
  }

  service { 'httpd':
    ensure  => 'running',
    enable  => true,
    require => Package['httpd'],
  }

}
```

2. Since the **rht-webtest** module has been updated, the metadata information for the module should also be updated. Edit the **metadata.json** file and increment the **version** field to 0.1.1. The resulting file should look like the following.

```
{
  "name": "rht-webtest",
  "version": "0.1.1",
  "author": "rht",
  "summary": "Red Hat Training test web server.",
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
  ]
}
```

3. Build an updated version of the module with the **puppet module build**.

```
[student@servera rht-webtest]$ cd ..
[student@servera labwork]$ puppet module build rht-webtest
Notice: Building /home/student/labwork/rht-webtest for release
Module built: /home/student/labwork/rht-webtest/pkg/rht-webtest-0.1.1.tar.gz
[student@servera labwork]$ ls rht-webtest/pkg
rht-webtest-0.1.0         rht-webtest-0.1.1
rht-webtest-0.1.0.tar.gz  rht-webtest-0.1.1.tar.gz
```

4. Add the files you changed to the staging area and commit them to the local Git repository with a helpful log message. Push them to the central Git repository.

```
[student@servera labwork]$ git add rht-webtest/manifests/init.pp
[student@servera labwork]$ git add rht-webtest/metadata.json
[student@servera labwork]$ git commit -m 'Created webtest Puppet class.'
[master 2982983] Created webtest Puppet class.
 2 files changed, 23 insertions(+), 19 deletions(-)
[student@servera labwork]$ git push
Counting objects: 11, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 775 bytes | 0 bytes/s, done.
Total 6 (delta 3), reused 0 (delta 0)
To ssh://workstation/var/git/labwork.git
   79fa396..2982983  master -> master
```

Deploying a Puppet Module Using a Smoke Test

Objectives

After completing this section, students should be able to:

- Deploy a Puppet module using a smoke test.

The **tests** directory in module hierarchy is for manifests that test the functionality provided by the Puppet manifests in the module. At a minimum, it should contain an **init.pp** smoke test manifest that corresponds with the **manifests/init.pp** manifest. Since **manifests/init.pp** defines the main class of the module, then **tests/init.pp** should simply use the class by including it with an **include** statement.

The **puppet module generate** command creates a working smoke test manifest based on the name of the module/class being defined. Compare the following template files created by the tool:

```
[root@host ~]# puppet module generate --skip-interview rht-modulename

Notice: Generating module at /root/rht-modulename...
Notice: Populating ERB templates...
Finished; module generated in rht-modulename.
... Output omitted ...
rht-modulename/manifests/init.pp
... Output omitted ...
rht-modulename/tests/init.pp
... Output omitted ...
[root@host ~]# tail rht-modulename/manifests/init.pp
# Author Name <author@domain.com>
#
# === Copyright
#
# Copyright 2015 Your name here, unless otherwise noted.
#
class modulename {


}
[root@host ~]# tail rht-modulename/tests/init.pp
# type.
#
# Tests are then run by using puppet apply --noop (to check for compilation
# errors and view a log of events) or by fully applying the test in a virtual
# environment (to compare the resulting system state to the desired state).
#
# Learn more about module testing here:
# http://docs.puppetlabs.com/guides/tests_smoke.html
#
include modulename
```

If the Puppet module creates new resource types or defines library functions that can be used by other modules, then additional files can be added to that directory to test more specific functionality. For Puppet classes with parameters, additional manifests can be defined that test the class with different parameter values.

Smoke test manifests can also show an administrator how to use the classes provided by a Puppet module. In this way they can serve as a minimal form of documentation.

Performing a smoke test

Using the **`puppet apply --noop`** command with the **`tests/init.pp`** manifest will display what commands and actions would be taken by Puppet when the class defined by the module is used. The **`Notice`** messages that print should be carefully studied to make sure all the necessary steps for proper system configuration would be taken.

Before a new module is published to a production environment, a more thorough smoke test can be performed by using the **`puppet apply`** without the **`--noop`** option on the **`tests/init.pp`** manifest. This should be performed on a fresh virtual machine so the changes made to the system configuration can be carefully studied.

References

For more information, see

> Module Smoke Testing
> http://docs.puppetlabs.com/guides/tests_smoke.html

Guided Exercise: Deploying a Puppet Module Using a Smoke Test

In this lab, you will modify a module and test its functionality by performing a smoke test.

Resources	
Files	**tests/init.pp**
Machines	**workstation** and **servera**

Outcome(s)
You should be able to write a Puppet module that confirms it produces the correct system state and test it.

Before you begin
The *puppet* package should already be installed on your **servera** host. You should also have a Puppet module from an earlier practice exercise that implements a web server with a Puppet class named **webtest**.

1. Log into **workstation** as **student** and inspect the test manifest for the **thoraxe-motd** Puppet module. Examine the **tests/init.pp** file.

```
[student@workstation ~]$ cat thoraxe-motd-0.1.1/tests/init.pp
# The baseline for module testing used by Puppet Labs is that each manifest
# should have a corresponding test manifest that declares that class or defined
# type.
#
# Tests are then run by using puppet apply --noop (to check for compilation
# errors and view a log of events) or by fully applying the test in a virtual
# environment (to compare the resulting system state to the desired state).
#
# Learn more about module testing here:
# http://docs.puppetlabs.com/guides/tests_smoke.html
#
include motd
```

   Since this module defines a Puppet class called **motd**, the test manifest simply includes the class to apply it.

2. Log into **servera** as **root** and inspect the test manifest for that module. It should include the Puppet class that was defined in **manifests/init.pp**.

```
[root@servera ~]# cd ~student/labwork/rht-webtest
[root@servera rht-webtest]# cat tests/init.pp
# The baseline for module testing used by Puppet Labs is that each manifest
# should have a corresponding test manifest that declares that class or defined
# type.
#
# Tests are then run by using puppet apply --noop (to check for compilation
# errors and view a log of events) or by fully applying the test in a virtual
# environment (to compare the resulting system state to the desired state).
#
# Learn more about module testing here:
# http://docs.puppetlabs.com/guides/tests_smoke.html
```

```
#
include webtest
```

Nothing needs to be done since the smoke test is already in place. The **puppet module generate** command created **tests/init.pp** with a reference to a module class named after the module name.

3. Since the module already had the smoke test in place, there is no need to increment the version, changing the metadata information. The module was already built in the previous practice exercise, so install it with the **puppet module install** command.

```
[root@servera rht-webtest]# ls -F /etc/puppet
auth.conf  modules/  puppet.conf
[root@servera rht-webtest]# puppet module install pkg/rht-webtest-0.1.1.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Created target directory /etc/puppet/modules
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── rht-webtest (v0.1.1)
[root@servera rht-webtest]# ls -F /etc/puppet
auth.conf  manifests/  modules/  puppet.conf
```

4. The **puppet module list** command will display a list of installed modules.

```
[root@servera rht-webtest]# puppet module list
/etc/puppet/modules
└── rht-webtest (v0.1.1)
/usr/share/puppet/modules (no modules installed)
```

5. Use **puppet apply --noop** on the test manifest to smoke test the Puppet class provided by the module. Even though the test manifest is in the module development directory, use the test manifest provided by the installed module in **/etc/puppet/modules/webtest/tests/init.pp**.

```
[root@servera rht-webtest]# puppet apply --noop /etc/puppet/modules/webtest/tests/
init.pp
Notice: Compiled catalog for servera.lab.example.com in environment production
 in 0.52 seconds
Warning: The package type's allow_virtual parameter will be changing its
 default value from false to true in a future release. If you do not want
 to allow virtual packages, please explicitly set allow_virtual to false.
   (at /usr/share/ruby/vendor_ruby/puppet/type.rb:816:in `set_default')
Notice: Finished catalog run in 0.11 seconds
```

6. The module would not make any changes to the system because all of the conditions of the desired state are already met. Remove the *httpd* package and apply the smoke test with the **--noop** option again.

```
[root@servera rht-webtest]# yum -y erase httpd
... Output omittted ...
[root@servera rht-webtest]# puppet apply --noop /etc/puppet/modules/webtest/tests/
init.pp
Notice: Compiled catalog for servera.lab.example.com in environment production
 in 0.53 seconds
Warning: The package type's allow_virtual parameter will be changing its
```

```
 default value from false to true in a future release. If you do not want
 to allow virtual packages, please explicitly set allow_virtual to false.
   (at /usr/share/ruby/vendor_ruby/puppet/type.rb:816:in `set_default')
Notice: /Stage[main]/Webtest/Package[httpd]/ensure: current_value absent, should be
 present (noop)
Notice: /Stage[main]/Webtest/Service[httpd]/ensure: current_value stopped, should be
 running (noop)
Notice: Class[Webtest]: Would have triggered 'refresh' from 2 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.21 seconds
```

7. The **--noop** option kept the **puppet apply** command from making changes to the system. A more extensive test could omit the option and have the changes applied to confirm the Puppet class works correctly.

```
[root@servera rht-webtest]# puppet apply /etc/puppet/modules/webtest/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment production
 in 0.55 seconds
Warning: The package type's allow_virtual parameter will be changing its
 default value from false to true in a future release. If you do not want
 to allow virtual packages, please explicitly set allow_virtual to false.
   (at /usr/share/ruby/vendor_ruby/puppet/type.rb:816:in `set_default')
Notice: /Stage[main]/Webtest/Package[httpd]/ensure: created
Notice: /Stage[main]/Webtest/Service[httpd]/ensure: ensure changed 'stopped' to
 'running'
Notice: Finished catalog run in 2.58 seconds
```

Lab: Implementing Puppet Modules

In this lab, you will take an existing Puppet manifest that configures an FTP server, change it into a Puppet class, and package it as a Puppet module.

Resources	
Files	**ftpserver.pp**
Machines	**servera**

Outcome(s)
You should be able to convert an existing Puppet manifest into a Puppet class and package it as a Puppet module. The module should not have any dependencies and should provide a smoke test that confirms the module configures the system correctly.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script. It creates the original Puppet manifest.

```
[student@workstation ~]$ lab puppet-modules setup
```

2. Log into **servera** as **root** and confirm the **ftpserver.pp** Puppet manifest is present.

```
[root@servera ~]# head -n 5 ftpserver.pp
package { 'vsftpd':
  ensure  => 'present',
}

user { 'ftpadmin':
```

3. Build a new directory hierarchy for a Puppet module called **rht-ftpserver**. Create it in the **labwork** directory so changes can be managed by Git.

4. Use the resources of the **ftpserver.pp** manifest to create the main Puppet class, **ftpserver**, of the **rht-ftpserver** Puppet module.

5. Add a smoke test for the Puppet module.

6. Since the module does not use library functions, modify it so that does not have any module dependencies.

7. Add the module source directory to the Git staging area. Commit the changes with a helpful log message.

8. Build the **rht-ftpserver** Puppet module.

9. Install the **rht-ftpserver** Puppet module.

10. Apply the module's smoke test and confirm it works. Remove the *vsftpd* package first so the module will take action to reinstall it.

11. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-modules grade
```

12. Commit any additional changes into the local Git repository. Once you have successfully finished the lab, pull (not fetch) upstream changes into the local repository, then push your recent changes to the upstream Git repository.

Solution

In this lab, you will take an existing Puppet manifest that configures an FTP server, change it into a Puppet class, and package it as a Puppet module.

Resources	
Files	**ftpserver.pp**
Machines	**servera**

Outcome(s)

You should be able to convert an existing Puppet manifest into a Puppet class and package it as a Puppet module. The module should not have any dependencies and should provide a smoke test that confirms the module configures the system correctly.

Before you begin

The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script. It creates the original Puppet manifest.

```
[student@workstation ~]$ lab puppet-modules setup
```

2. Log into **servera** as **root** and confirm the **ftpserver.pp** Puppet manifest is present.

```
[root@servera ~]# head -n 5 ftpserver.pp
package { 'vsftpd':
  ensure  => 'present',
}

user { 'ftpadmin':
```

3. Build a new directory hierarchy for a Puppet module called **rht-ftpserver**. Create it in the **labwork** directory so changes can be managed by Git.

   Use the **puppet module generate** command to create the directory hierarchy and Puppet module metadata for you. You can accept default values for much of the metadata.

```
[root@servera ~]# cd labwork
[root@servera labwork]# puppet module generate rht-ftpserver
We need to create a metadata.json file for this module.  Please answer the
following questions; if the question is not applicable to this module, feel free
to leave it blank.

Puppet uses Semantic Versioning (semver.org) to version modules.
What version is this module?  [0.1.0]
--> Enter

Who wrote this module?  [rht]
--> Enter

What license does this module code fall under?  [Apache 2.0]
--> Enter

How would you describe this module in a single sentence?
--> Deploy an ftp server with an administrative user.
```

```
Where is this module's source code repository?
--> Enter

Where can others go to learn more about this module?
--> Enter

Where can others go to file issues about this module?
--> Enter

----------------------------------------
{
  "name": "rht-ftpserver",
  "version": "0.1.0",
  "author": "rht",
  "summary": "Deploy an ftp server with an administrative user.",

... Output omitted ...

About to generate this metadata; continue? [n/Y]
--> Enter
... Output omitted ...
```

4. Use the resources of the **ftpserver.pp** manifest to create the main Puppet class, **ftpserver**, of the **rht-ftpserver** Puppet module.

   The **manifests/init.pp** already has definition for an empty **ftpserver** Puppet class. Insert the resource records from **ftpserver.pp** inside of the class definition.

```
[root@servera labwork]# cd rht-ftpserver
[root@servera rht-ftpserver]# vi manifests/init.pp
[root@servera rht-ftpserver]# cat manifests/init.pp
# == Class: ftpserver
#
# Full description of class ftpserver here.
#
... Output omitted ...
#
# === Copyright
#
# Copyright 2015 Your name here, unless otherwise noted.
#
class ftpserver {

  package { 'vsftpd':
    ensure  => 'present',
  }
... Output omitted ...
}
```

5. Add a smoke test for the Puppet module.

   If **puppet module generate** was used, then the smoke test will already be created; otherwise, create a file in the directory hierarchy, called **tests/init.pp**, that contains content similar to the following.

```
[root@servera rht-ftpserver]# cat tests/init.pp
... Output omitted ...
#
```

```
include ftpserver
```

6. Since the module does not use library functions, modify it so that does not have any module dependencies.

   Edit the module's **metadata.json** file to remove the default **puppetlabs-stdlib** dependency. The resulting file should be similar to the following.

```
{
  "name": "rht-ftpserver",
  "version": "0.1.0",
  "author": "rht",
  "summary": "Deploy an ftp server with an administrative user.",
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
  ]
}
```

7. Add the module source directory to the Git staging area. Commit the changes with a helpful log message.

```
[root@servera rht-ftpserver]# git add .
[root@servera rht-ftpserver]# git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README.md
#       new file:   Rakefile
#       new file:   manifests/init.pp
#       new file:   metadata.json
#       new file:   spec/classes/init_spec.rb
#       new file:   spec/spec_helper.rb
#       new file:   tests/init.pp
#
[root@servera rht-ftpserver]# git commit -m 'Initial version of rht-ftpserver.'
[master 2716e16] Initial version of rht-ftpserver.
 7 files changed, 217 insertions(+)
 create mode 100644 rht-ftpserver/README.md
 create mode 100644 rht-ftpserver/Rakefile
 create mode 100644 rht-ftpserver/manifests/init.pp
 create mode 100644 rht-ftpserver/metadata.json
 create mode 100644 rht-ftpserver/spec/classes/init_spec.rb
 create mode 100644 rht-ftpserver/spec/spec_helper.rb
 create mode 100644 rht-ftpserver/tests/init.pp
```

8. Build the **rht-ftpserver** Puppet module.

   Use the **puppet module build** command to build the Puppet module.

```
[root@servera rht-ftpserver]# cd ..
[root@servera labwork]# puppet module build rht-ftpserver
```

9. Install the **rht-ftpserver** Puppet module.

Use the **puppet module install** command to install the Puppet module.

```
[root@servera labwork]# puppet module install rht-ftpserver/pkg/rht-
ftpserver-0.1.0.tar.gz
```

10. Apply the module's smoke test and confirm it works. Remove the *vsftpd* package first so the module will take action to reinstall it.

    Use the **puppet apply --noop** on the **tests/init.pp** file in the installed module's directory to test the Puppet module.

```
[root@servera labwork]# yum -y erase vsftpd
... Output omitted ...
[root@servera labwork]# puppet apply --noop /etc/puppet/modules/ftpserver/tests/
init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.49 seconds
... Output omitted ...
Notice: /Stage[main]/Ftpserver/Package[vsftpd]/ensure: current_value absent, should
 be present (noop)
Notice: /Stage[main]/Ftpserver/Service[vsftpd]/ensure: current_value stopped, should
 be running (noop)
Notice: /Stage[main]/Ftpserver/File[/var/ftp]/ensure: current_value absent, should
 be directory (noop)
Notice: /Stage[main]/Ftpserver/File[/var/ftp/pub]/ensure: current_value absent,
 should be directory (noop)
Notice: Class[Ftpserver]: Would have triggered 'refresh' from 4 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.42 seconds
```

11. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-modules grade
```

12. Commit any additional changes into the local Git repository. Once you have successfully finished the lab, pull (not fetch) upstream changes into the local repository, then push your recent changes to the upstream Git repository.

```
[root@servera labwork]# git pull
Merge made by the 'recursive' strategy.
 rht-webtest/README.md                | 79 ++++++++++++++++++++++++++++++++++
 rht-webtest/Rakefile                 | 18 ++++++++
 rht-webtest/manifests/init.pp        | 64 ++++++++++++++++++++++++++++
 rht-webtest/metadata.json            | 13 ++++++
 rht-webtest/spec/classes/init_spec.rb |  7 ++++
 rht-webtest/spec/spec_helper.rb      | 17 ++++++++
 rht-webtest/tests/init.pp            | 12 ++++++
 7 files changed, 210 insertions(+)
... Output omitted ...
[root@servera labwork]# git push
Counting objects: 16, done.
Compressing objects: 100% (11/11), done.
Writing objects: 100% (14/14), 3.53 KiB | 0 bytes/s, done.
Total 14 (delta 2), reused 0 (delta 0)
To ssh://student@workstation/var/git/labwork.git
```

```
   2982983..689b34d  master -> master
```

Summary

In this chapter, you learned:

- The **`puppet module generate`** command will create a template directory structure for developing a Puppet module.
- The **`puppet module build`** command takes a module development directory and assembles it into a module.
- The **`puppet module install`** command extracts a Puppet module into a directory under **`/etc/puppet/modules`**.
- Puppet modules should define Puppet classes, one per manifest, and the main class defined in **`manifests/init.pp`** should have the same name as the module.
- Every Puppet module should have a smoke test defined in **`tests/init.pp`** that includes the main class defined by the module.

CHAPTER 8

IMPLEMENTING RELATIONSHIPS IN A PUPPET MODULE

Overview	
Goal	Implement namespaces, relationships, and dependencies in a Puppet module.
Objectives	• Create namespaces in a Puppet module. • Create relationships and dependencies in a Puppet module. • Reference a file in a Puppet module.
Sections	• Creating Namespaces (and Guided Exercise) • Creating Relationships and Dependencies (and Guided Exercise) • Referencing Files (and Guided Exercise)
Lab	• Implementing Relationships in a Puppet Module

Creating Namespaces

Objectives

After completing this section, students should be able to:

- Create namespaces in a Puppet module.

The main manifest in a Puppet module is the **manifests/init.pp**. This manifest defines the class that has the same name as the module. When this manifest gets very large, sometimes it is useful to break the class up into smaller classes based on functionality: perhaps **file** resources, **user** resources, and **service** resources. Optional resources can be broken out into separate classes. It is possible to group resources into separate classes based on their function in the deployment of a service: install, config, service.

When a class is broken up into related classes, the class names and the manifests that define them matter. They get broken up into segments called namespaces. Namespaces tell the Puppet autoloader how to find specific classes that are defined in a module.

Mapping namespaces in modules

A good way to get started with namespaces is to look at a working example of them being used. Many of the manifests provided by the **ssh** module for the OpenStack installer define classes. The relevant manifests are found in the **/usr/share/openstack-puppet/modules/ssh/manifests** directory. The following output from the **tree** command shows the list of manifests.

```
[root@host manifests]# tree
.
├── client
│   ├── config
│   │   └── user.pp
│   ├── config.pp
│   └── install.pp
├── client.pp
├── hostkeys.pp
├── init.pp
├── knownhosts.pp
├── params.pp
├── server
│   ├── config.pp
│   ├── host_key.pp
│   ├── install.pp
│   ├── match_block.pp
│   └── service.pp
└── server.pp

3 directories, 14 files
```

Recursively searching for the string "class" identifies which manifests define another class in the namespace of the module. The namespace of the module is **ssh** and the **init.pp** manifest defines the **ssh** class as expected.

```
[root@host manifests]# grep -R 'class' .
./client/config.pp:class ssh::client::config
./client/install.pp:class ssh::client::install {
```

```
./client.pp:class ssh::client(
./hostkeys.pp:class ssh::hostkeys {
./init.pp:class ssh (
./init.pp: class { 'ssh::server':
./init.pp: class { 'ssh::client':
./knownhosts.pp:class ssh::knownhosts {
./params.pp:class ssh::params {
./server/config.pp:class ssh::server::config {
./server/install.pp:class ssh::server::install {
./server/service.pp:class ssh::server::service {
./server.pp:class ssh::server(
```

Note how the **client.pp** manifest defines the **ssh::client** class. The first element of the name, **ssh**, is the module's namespace, and **client** is the name of the class being defined in that namespace. This class is being referenced within the **ssh** class definition in **init.pp**. The same is true for the **ssh::server** class, which is defined in the **server.pp** file.

The **install.pp** manifest found in the **server** subdirectory defines a class called **ssh::server::install**. As before, **ssh** is the name of the module, but **install** is the name of the class being defined in the **ssh::server** namespace. The namespace tells the Puppet autoloader the module that defines the object, and it tells it where to find the definition in the directory structure of the module.

The double colon (::) separates the elements of the namespace in an object's name. It is similar to a full pathname for a Puppet object. When a simple name is used, a single element, the double colon separator is not needed and the reference applies to a Puppet object that is defined in the current manifest.

Creating additional smoke tests

With more complex modules, additional smoke tests can be written to test the functionality of the new classes. A good practice to adopt is to create manifests in **tests** that map to the corresponding manifest in **manifests** that they exercise.

The **tests** directory of the OpenStack SSH Puppet module has two smoke test manifests: **init.pp** and **server.pp** The following is the content of **server.pp**.

```
include ssh::server
```

It is not necessary to define a smoke test for every class defined in a module, but some additional smoke tests can be useful to test a subset of a module's functionality.

References

For more information, see

Puppet Labs - Language: Namespaces and Autoloading
http://docs.puppetlabs.com/puppet/3.6/reference/lang_namespaces.html

Guided Exercise: Creating Namespaces

In this lab, you will modify the **webtest** module to make it more modular. You will create and use classes for creating files and users, in addition to deploying the service.

Resources	
Files	**/var/www/html/index.html**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet class that uses other classes and packages them into a module.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host. You should also have a Puppet module from an earlier practice exercise that implements a web server with a Puppet class named **webtest**.

1. Log in as **student** on **servera** and change into the **labwork/rht-webtest** directory and investigate the module. It originally should have a single manifest in the **manifests** directory, **init.pp**.

```
[student@servera ~]$ cd labwork/rht-webtest
[student@servera rht-webtest]$ ls
manifests  metadata.json  pkg  Rakefile  README.md  spec  tests
[student@servera rht-webtest]$ ls manifests
init.pp
```

2. Pull any upstream changes into the local Git repository and working tree.

```
[student@servera rht-webtest]$ git pull
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (14/14), done.
From ssh://workstation/var/git/labwork
   2982983..689b34d  master     -> origin/master
... Output omitted ...
```

3. Create a new class, called **rht-webtest::files**, that manages the files managed by the **rht-webtest** class.

   3.1. Copy the original **manifests/init.pp** file to **manifests/files.pp** to recycle the **file** resources defined in it.

```
[student@servera rht-webtest]$ cp manifests/init.pp manifests/files.pp
[student@servera rht-webtest]$ ls manifests
files.pp  init.pp
```

   3.2. Edit **manifests/files.pp** so it defines a class, named **webtest::files**. When you are finished, it should look like the following.

```
# == Class: webtest::files
#

class webtest::files {

  file { '/var/www/html/index.html':
    ensure  => 'file',
    mode    => '0644',
    content => "<h1>This web site managed by Puppet.</h1>\n",
    require => Package['httpd'],
  }

}
```

4. Create two new classes, called **webtest::users::alice** and **webtest::users::bob**, that manage user accounts for web administrators.

   4.1. Create a directory in the module hierarchy called **manifests/users**. This will contain the manifests that define the classes in the **webtest::users** namespace.

```
[student@servera rht-webtest]$ mkdir manifests/users
```

   4.2. Create a manifest that defines the class that manages the **alice** user account. The manifest should be called **manifests/users/alice.pp** and it should define a class named **webtest::users::alice**. When you are finished, it should look like the following:

```
# == Class: webtest::users::alice
#

class webtest::users::alice {

  user { 'alice':
    ensure  => 'present',
    groups  => 'webmaster',
    home    => '/home/webmaster',
    shell   => '/bin/bash',
    require => File['/home/webmaster'],
  }

}
```

   4.3. Do the same for the class that manages the **bob** user account. When you are finished, the manifest named **manifests/users/bob.pp** should look like the following:

```
# == Class: webtest::users::bob
#

class webtest::users::bob {

  user { 'bob':
    ensure  => 'present',
    groups  => 'webmaster',
    home    => '/home/webmaster',
    shell   => '/bin/bash',
    require => File['/home/webmaster'],
```

```
  }

}
```

5. Edit the main module manifest, **init.pp**. Remove any resource definitions that are managed by the new classes that have been defined. Include the new classes in the main class. When you are finished, **init.pp** should look like the following.

```
# == Class: webtest
#
... Output omitted ...

class webtest {

  user { 'webmaster':
    ensure  => 'present',
    home    => '/home/webmaster',
    shell   => '/bin/bash',
  }

  file { '/home/webmaster':
    ensure  => 'directory',
    owner   => 'webmaster',
    mode    => '0770',
    require => User['webmaster'],
  }

  package { 'httpd':
    ensure  => 'present',
  }

  service { 'httpd':
    ensure  => 'running',
    enable  => true,
    require => Package['httpd'],
  }

  include 'webtest::files'
  include 'webtest::users::alice'
  include 'webtest::users::bob'

}
```

6. Update the **metadata.json** file and increment the module version.

```
{
  "name": "rht-webtest",
  "version": "0.1.2",
  "author": "rht",
  "summary": "Red Hat Training test web server.",
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
  ]
}
```

7. Add the manifest and **metadata.json** changes to the Git staging area. Commit the changes to the local Git repository with a descriptive log message.

```
[student@servera rht-webtest]$ git add manifests metadata.json
[student@servera rht-webtest]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   manifests/files.pp
#       modified:   manifests/init.pp
#       new file:   manifests/users/alice.pp
#       new file:   manifests/users/bob.pp
#       modified:   metadata.json
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       pkg/
[student@servera rht-webtest]$ git commit -m 'Created more namespaces.'
[master 0e5e0ac] Created more namespaces.
 5 files changed, 46 insertions(+), 1 deletion(-)
 create mode 100644 rht-webtest/manifests/files.pp
 create mode 100644 rht-webtest/manifests/users/alice.pp
 create mode 100644 rht-webtest/manifests/users/bob.pp
```

8. Build the **rht-webtest** module.

```
[student@servera rht-webtest]$ cd ..
[student@servera labwork]$ puppet module build rht-webtest
Notice: Building /home/student/labwork/rht-webtest for release
Module built: /home/student/labwork/rht-webtest/pkg/rht-webtest-0.1.2.tar.gz
```

9. Log in as **root** on **servera** (password is **redhat**). Install the **rht-webtest** module.

```
[root@servera ~]# puppet module install --force \
      ~student/labwork/rht-webtest/pkg/rht-webtest-0.1.2.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── rht-webtest (v0.1.2)
```

10. Apply the installed module smoke test with the **--noop** option to confirm the actions it intends to take.

```
[root@servera ~]# puppet apply --noop /etc/puppet/modules/webtest/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.03 seconds
... Output omitted ...
Notice: /Stage[main]/Webtest::Users::Bob/User[bob]/ensure: current_value
 absent, should be present (noop)
Notice: Class[Webtest::Users::Bob]: Would have triggered 'refresh'
 from 1 events
Notice: /Stage[main]/Webtest::Files/File[/var/www/html/index.html]/ensure:
 current_value absent, should be file (noop)
Notice: Class[Webtest::Files]: Would have triggered 'refresh' from 1 events
Notice: /Stage[main]/Webtest::Users::Alice/User[alice]/ensure:
```

```
 current_value absent, should be present (noop)
Notice: Class[Webtest::Users::Alice]: Would have triggered 'refresh'
 from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 3 events
Notice: Finished catalog run in 0.39 seconds
```

11. Apply the module smoke test and confirm the files and users are present.

    11.1. First, apply the installed module smoke test.

```
[root@servera ~]# puppet apply /etc/puppet/modules/webtest/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.01 seconds
... Output omitted ...
Notice: /Stage[main]/Webtest::Users::Bob/User[bob]/ensure: created
Notice: /Stage[main]/Webtest::Files/File[/var/www/html/index.html]/ensure:
 defined content as '{md5}6e9ca937d95e30d6a4b04a2053860cf6'
Notice: /Stage[main]/Webtest::Users::Alice/User[alice]/ensure: created
Notice: Finished catalog run in 0.58 seconds
```

    11.2. The **alice** and **bob** users should be present.

```
[root@servera ~]# id alice
uid=1013(alice) gid=1013(alice) groups=1013(alice),1001(webmaster)
[root@servera ~]# id bob
uid=1012(bob) gid=1012(bob) groups=1012(bob),1001(webmaster)
```

    11.3. The **index.html** file should be created with the correct contents.

```
[root@servera ~]# ls -l /var/www/html
total 4
-rw-r- -r- -. 1 root root 42 Sep 23 16:27 index.html
[root@servera ~]# cat /var/www/html/index.html
<h1>This web site managed by Puppet.</h1>
```

12. As **student**, commit any additional changes you made into the local Git repository. Once you have successfully finished the exercise, push your committed changes to the upstream Git repository.

```
[student@servera labwork]$ git push
... Output omitted ...
```

Creating Relationships and Dependencies

Objectives

After completing this section, students should be able to:

- Create relationships and dependencies in a Puppet module.

Defining relationships between Puppet objects

Puppet manifests describe the desired end state of a system. They are not procedural, top-down structured programs. Puppet is free to process the resources in a manifest in any order that it considers best. Neither the order nor the type of resources imply any order that they should be accomplished. If a group of resources should be implemented in a specific order, then the relationships between them must explicitly be declared.

Using relationship metaparameters

One way to declare the relationship between resources is with relationship metaparameters. The Puppet language provides four relationship metaparameters that can be used with any resource type. They are the **before**, **require**, **notify**, and **subscribe** metaparameters.

These metaparameters are used in the body of resource definitions. Their values are references to other Puppet objects, which have the following form:

```
Resource_type[ resource_title ]
```

The *Resource_type* must be capitalized when used in this way. Usually a single resource title is specified in the square brackets, but a comma-separated list of titles can be used when a reference to multiple resources is desired.

Consider the following example:

```
file { '/home/webmaster':
  ensure  => 'directory',
  owner   => 'webmaster',
  mode    => '0770',
  require => User['webmaster'],
}
```

The resource that defines the **webmaster** user must be applied before the **file** resource that manages the **/home/webmaster** directory is created.

An alternate way to specify this relationship between the resources is to use the **before** metaparameter in the user resource. The resulting **user** resource would look like the following:

```
user { 'webmaster':
  ensure  => 'present',
  home    => '/home/webmaster',
  shell   => '/bin/bash',
  before  => File['/home/webmaster'],
}
```

Either the **require** or **before** metaparameter can be used; it is not necessary to specify both.

The **subscribe** and **notify** metaparameters are similar to the **require** and **before** metaparameters, except a notification is sent to the dependent resource to "refresh" that resource. A simple example of this relationship occurs when a configuration file for a service is modified.

```
service { 'webserver':
  ensure  => 'running',
  name    => 'httpd',
  enable  => true,
}

file { '/etc/httpd/conf/httpd.conf':
  ensure  => 'file',
  owner   => 'root',
  group   => 'root',
  notify  => Service['webserver'],
  ... Output omitted ...
}
```

In the previous example, the **httpd.conf** file is created by Puppet before starting the **httpd** service. In terms of resource ordering, the **notify** metaparameter is similar to **before**. Where it differs is that when the configuration file is modified, the **httpd** service is refreshed. Usually this results in the service being restarted on the host.

The following example is equivalent to the previous example, except it uses the **subscribe** metaparameter instead of **notify**.

```
service { 'webserver':
  ensure    => 'running',
  name      => 'httpd',
  enable    => true,
  subscribe => File['/etc/httpd/conf/httpd.conf'],
}

file { '/etc/httpd/conf/httpd.conf':
  ensure  => 'file',
  owner   => 'root',
  group   => 'root',
  ... Output omitted ...
}
```

Referencing resources: title vs. namevar

Every resource must have a title specified. The title is the first string in the resource declaration, immediately followed by a colon. Every resource type has a parameter that specifies the local name of the resource; this is referred to as the *namevar* because it is usually the **name** parameter.

The **name** parameter is the namevar of the **service** resource. It specifies the name of the service to be managed. The namevar for the **file** resource is the **path** parameter.

When the namevar of a resource is omitted, Puppet uses the title of the resource for the value of the implied namevar. The following example illustrates this case. Although the **path** parameter is omitted, the path name of the directory to be managed is **/var/www/html/examples**, the title.

```
file { '/var/www/html/examples':
  ensure  => 'directory',
  owner   => 'root',
```

```
  mode    => '0755',
}
```

The titles of Puppet objects are used to reference them in Puppet DSL. This is useful for Puppet objects, such as services, that may refer to different service names based on the operating system in use. For example, the NTP service would be defined the following way on Red Hat Enterprise Linux 6:

```
service { 'ntp':
  name    => 'ntpd',
  ensure  => 'running',
  enable  => true,
}
```

On a Red Hat Enterprise Linux 7 system this resource would look like the following instead:

```
service { 'ntp':
  name    => 'chrony',
  ensure  => 'running',
  enable  => true,
}
```

Since the title is used in references, both of the previous examples would be properly referred to as **`Service['ntp']`**. The title, **`ntp`**, is the name that the Puppet parser uses to resolve the reference.

Using timelines to express relationships

Timelines are another way to express relationships between Puppet objects. They are defined by the **`->`** and **`~>`** operators. When a timeline is used, the order of the resource definitions matters, they are processed from left to right.

The **`->`** operator declares an ordered relationship. The resource on the left of the operator must be managed before the resource on the right is processed. The **`~>`** defines the same ordering, except that a notification relationship is also established. If the resource on the left changes, then the resource on the right of this operator is refreshed.

The following is a code snippet taken from the **`ntp/manifests/init.pp`** manifest that is included with the OpenStack installer. It demonstrates the use of a timeline for defining dependencies, in this case between Puppet classes.

```
anchor { 'ntp::begin': } ->
class { '::ntp::install': } ->
class { '::ntp::config': } ~>
class { '::ntp::service': } ->
anchor { 'ntp::end': }
```

The resources of the **`::ntp::install`** class are applied first. Once they have been processed, the resources of the **`::ntp::config`** class are applied. Finally the resources of the **`::ntp::service`** class are applied. Since the **`~>`** timeline operator was used between the **`::ntp::config`** and **`::ntp::service`** classes, any resource updates to the configuration files will cause the services in the second class to be refreshed.

The **`class`** relationships are enclosed within the **`anchor`** patterns for backward compatibility with Puppet, version 3.4 and earlier. This construct causes the three classes defined within the

anchors to be contained by the class in which they are used, so they become private to that class and cannot be referenced by other classes.

Troubleshooting failed dependencies

When a reference is made to a nonexistent resource, this missing dependency will cause a compilation error. Puppet will not apply any of the resources in a manifest when this condition occurs.

If Puppet cannot apply a prerequisite resource, then it will not continue with the resources that depend on the prerequisite. Unrelated resources will continue to be processed. When this condition occurs, the following warning will be issued:

```
warning: RESOURCE: Skipping because of failed dependencies
```

Another dependency issue to watch out for is a circular dependency. This is when the dependencies between two or more resources results in a loop. Puppet will compile the manifest, but since it cannot apply it, the following error message will display:

```
err: Could not apply complete catalog: Found 1 dependency cycle:
(RESOURCE => OTHER RESOURCE => RESOURCE)
```

References

For more information, see

> Puppet Labs - Language: Relationships and Ordering
> http://docs.puppetlabs.com/puppet/3.6/reference/lang_relationships.html

For more information, see

> Puppet Labs - Metaparameter Reference
> http://docs.puppetlabs.com/references/3.6.latest/metaparameter.html

Guided Exercise: Creating Relationships and Dependencies

In this lab, you will use the **require** and **before** metaparameters to define a relationship between related resources in a Puppet manifest.

Resources	
Machines	**servera**

Outcome(s)
You should be able to properly define relationships between related resources in a Puppet manifest.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log in as **root** on **servera**. Change into the **labwork** directory so any work you do there can be managed with Git. Before gettin started, pull all upstream changes into the local Git repository and working tree.

```
[root@servera ~]# cd labwork
[root@servera labwork]# git pull
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (10/10), done.
... Output omitted ...
```

2. Create a manifest, called **relate-do.pp**, that installs the *httpd* package. It should also start and configure the **httpd** service to start at boot time. The resulting manifest should look like the following.

```
package { 'httpd':
  ensure  => 'present',
}

service { 'httpd':
  ensure  => 'running',
  enable  => true,
}
```

3. Create another manifest, called **relate-undo.pp**, that stops and disables the **httpd** service. It should also remove the *httpd* package. The undo manifest should look similar to the following.

```
package { 'httpd':
  ensure  => 'absent',
}

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

4. Add and commit the two new manifests to the local Git repository.

```
[root@servera labwork]# git add relate-*.pp
[root@servera labwork]# git commit -m 'Initial versions.'
[master af45e2b] Initial versions.
 2 files changed, 20 insertions(+)
 create mode 100644 relate-do.pp
 create mode 100644 relate-undo.pp
```

5. Apply the **relate-undo.pp** manifest. It will reset the system to a known state where the **httpd** service is not running and the *httpd* package is not installed.

```
[root@servera ~]# puppet apply relate-undo.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.84 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Service[httpd]/ensure: ensure changed 'running'
 to 'stopped'
Notice: /Stage[main]/Main/Package[httpd]/ensure: removed
Notice: Finished catalog run in 2.18 seconds
```

6. Apply the **relate-do.pp** manifest to install the *httpd* package and start the **httpd** service. It may display a **systemctl** error if it tries to start the service before the package is installed.

```
[root@servera ~]# puppet apply relate-do.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.86 seconds
... Output omitted ...
Error: Could not start Service[httpd]: Execution of '/usr/bin/systemctl
 start httpd' returned 6: Failed to issue method call: Unit httpd.service
 failed to load: No such file or directory.
Wrapped exception:
Execution of '/usr/bin/systemctl start httpd' returned 6: Failed to issue
 method call: Unit httpd.service failed to load: No such file or directory.
Error: /Stage[main]/Main/Service[httpd]/ensure: change from stopped
 to running failed: Could not start Service[httpd]: Execution of
 '/usr/bin/systemctl start httpd' returned 6: Failed to issue method call:
 Unit httpd.service failed to load: No such file or directory.
Notice: /Stage[main]/Main/Package[httpd]/ensure: created
Notice: Finished catalog run in 5.22 seconds
```

7. Restore the system to a known state. Use **puppet apply** to disable the service and remove the package.

```
[root@servera ~]# puppet apply relate-undo.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.83 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Package[httpd]/ensure: removed
Notice: Finished catalog run in 0.76 seconds
```

8. Edit the **relate-do.pp** manifest and add a **require** metaparameter to the **service** resource. It should require that the *httpd* package be installed first.

```
package { 'httpd':
  ensure  => 'present',
}

service { 'httpd':
  ensure  => 'running',
  enable  => true,
  require => Package['httpd'],
}
```

9. Commit the change to **relate-do.pp** to the local Git repository. Provide a meaningful log message when it is committed.

```
[root@servera labwork]# git add relate-do.pp
[root@servera labwork]# git commit -m 'Add require metaparameter to service.'
... Output omitted ...
```

10. Apply the updated **relate-do.pp** manifest. Since it defines the appropriate relationship between the service and the package resources, the package is installed first, then the service is started by Puppet.

```
[root@servera ~]# puppet apply relate-do.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.87 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Package[httpd]/ensure: created
Notice: /Stage[main]/Main/Service[httpd]/ensure: ensure changed 'stopped'
 to 'running'
Notice: Finished catalog run in 7.84 seconds
```

11. Use Puppet to disable the service and remove the package.

```
[root@servera ~]# puppet apply relate-undo.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.85 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Service[httpd]/ensure: ensure changed 'running'
 to 'stopped'
Notice: /Stage[main]/Main/Package[httpd]/ensure: removed
Notice: Finished catalog run in 2.10 seconds
```

12. Another way of expressing the relationship between the **package** and the **service** resources is that the *httpd* package must be installed before the **httpd** service resource can be managed. Remove the **require** metaparameter from the **service** resource and add a **before** metaparameter to the **package** resource instead.

```
package { 'httpd':
  ensure  => 'present',
  before => Service['httpd'],
}

service { 'httpd':
  ensure  => 'running',
  enable  => true,
```

```
}
```

13. Commit the metaparameter change to **relate-do.pp** to the local Git repository. Provide a meaningful log message when it is committed.

```
[root@servera labwork]# git add relate-do.pp
[root@servera labwork]# git commit -m 'Use before metaparameter instead of require.'
... Output omitted ...
```

14. Apply the updated **relate-do.pp** manifest. Since it also defines the appropriate relationship between the service and the package resources, the package is installed first, then the service is started by Puppet.

```
[root@servera ~]# puppet apply relate-do.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.70 seconds
... Output omitted ...
Notice: /Stage[main]/Main/Package[httpd]/ensure: created
Notice: /Stage[main]/Main/Service[httpd]/ensure: ensure changed 'stopped'
 to 'running'
Notice: Finished catalog run in 8.00 seconds
```

15. Commit any additional changes you made into the local Git repository. Once you have successfully finished the exercise, push your committed changes to the upstream Git repository.

```
[root@servera labwork]# git push
... Output omitted ...
```

Referencing Files

Objectives

After completing this section, students should be able to:

- Reference a file in a Puppet module.

Providing file content from Puppet modules

Puppet developers use the **file** resource to define the state of a file on the file system. The **content** parameter of this resource takes a string that determines the file's contents. If there is much textual material, it would clutter the manifest with volumes of content. This section shows a couple of Puppet alternatives for providing copious file content by modules.

Providing static file content

When static file content must be provided where nothing needs to be changed, the **source** parameter can be used in the **file** resource. Note that it is mutually exclusive with the **content** parameter; either one or the other can be used in a given **file** resource.

The value of the **source** parameter should be a pathname to a file provided by the module. The following shows how the **source** parameter is used:

```
source => 'puppet:///modules/MODULENAME/FILENAME',
```

MODULENAME is the name of the Puppet module that provides the file. *FILENAME* is the name of the original file being provided. The file is published by putting it in the **files** directory when writing the Puppet module. This directory must be at the same level as the **manifests** directory in the Puppet module development tree.

To publish a file called **widget.conf** from a Puppet module called **acme**, the original **widget.conf** would be copied to the **rht-acme/files** directory of the module. The related **file** resource in the module manifests would look like the following:

```
source => 'puppet:///modules/acme/widget.conf',
```

Providing variable file content

Templates are another method for providing large amounts of file content to **file** resources. The manifest that uses templates defines the **file** resource with a **content** parameter, but the value is a call to the built-in **template()** function.

```
content => template('MODULENAME/FILENAME.erb'),
```

Puppet looks for the template in a file called ***FILENAME*.erb** in the **templates** directory of the module referenced, in this case ***MODULENAME***.

Embedded Ruby (ERB) is the language that Puppet uses for templates. Its expressions are enclosed within **<% %>** tags. The following are a couple of the simplest ERB constructs:

```
<%# COMMENT %>
```

```
<%= EXPRESSION %>
```

The first form is an ERB comment that is used to document the template. It does not change the file content in any way. The second form modifies the content of the template file. It inserts the value of the enclosed *EXPRESSION*, often a system fact, into the text.

The following template, **hosts.erb**, could be used with the Puppet **template()** function to create content for **/etc/hosts** in Linux.

```
127.0.0.1   localhost localhost.localdomain
::1         localhost localhost.localdomain

<%= @ipaddress %>   <%= @fqdn %>
```

The first expression in the template expands to the client host's IP address and the second expression expands to the fully qualified host name. The @ prefix tells ERB to use an external variable or fact, rather than one of its own variables.

The following is a useful Bash function for checking the syntax of an ERB template. It is a useful function to define in the **.bashrc** of a Puppet developer, or its code can be used in a Git hook for checking the syntax of a template before committing it into a repository.

```
validate_erb() {
  erb -P -x -T - $1 | ruby -c
}
```

References

For more information, see

> Puppet Labs - Type Reference: file
> http://docs.puppetlabs.com/references/3.6.latest/type.html#file

Guided Exercise: Referencing Files

In this lab, you will modify the **webtest** module to make it use provided files or templates for a **file** resource.

Resources	
Files	**/var/www/html/index.html**
Machines	**servera**

Outcome(s)
You should be able to build a Puppet module that provides fixed file content or a template and make reference to them in **file** resources so they are deployed.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host. You should also have a Puppet module from an earlier practice exercise that implements a web server with a Puppet class named **webtest**.

1. Log in as **student** on **servera**. Change into the **labwork** directory and pull any updates from the central Git repository.

```
[student@servera ~]$ cd labwork
[student@servera labwork]$ git pull
... Output omitted ...
```

2. Change into the working directory for **rht-webtest** and create a static file to be deployed as **index.html**. You will need to create the **files** directory in the module directory hierarchy.

```
[student@servera labwork]$ cd rht-webtest
[student@servera rht-webtest]$ mkdir files
[student@servera rht-webtest]$ vim files/index.html
[student@servera rht-webtest]$ cat files/index.html
<h1>Puppet provided index.html</h1>
<p>
  This is a file provided in the files/ directory.
</p>
```

3. Modify the **files.pp** manifest so the contents of the **index.html file** resource come from the file provided by the Puppet module, instead of being hard-coded into the resource Puppet DSL.

```
[student@servera rht-webtest]$ vi manifests/files.pp
[student@servera rht-webtest]$ cat manifests/files.pp
# == Class: webtest::files
#

class webtest::files {

  file { '/var/www/html/index.html':
    ensure  => 'file',
    mode    => '0644',
```

```
    source => 'puppet:///modules/webtest/index.html',
    require => Package['httpd'],
  }

}
```

This requires use of **source** instead of the **content** parameter. The value tells Puppet where to get the file from, namely the **webtest** Puppet module under the name **index.html**.

4. Increment the module version in the metadata file.

```
[student@servera rht-webtest]$ vim metadata.json
[student@servera rht-webtest]$ grep version metadata.json
  "version": "0.1.3",
```

5. Add the **files** directory and the manifest and **metadata.json** changes to the Git staging area. Commit the changes to the local Git repository with a descriptive log message.

```
[student@servera rht-webtest]$ git add files manifests/files.pp metadata.json
[student@servera rht-webtest]$ git commit -m 'Put index.html content in a file.'
... Output omitted ...
```

6. Build a new version of the module.

```
[student@servera rht-webtest]$ cd ..
[student@servera labwork]$ puppet module build rht-webtest
Notice: Building /home/student/labwork/rht-webtest for release
Module built: /home/student/labwork/rht-webtest/pkg/rht-webtest-0.1.3.tar.gz
```

7. Log in as **root** on **servera** and install the new version of the module.

```
[root@servera ~]# puppet module uninstall --force rht-webtest
Notice: Preparing to uninstall 'rht-webtest' ...
Removed 'rht-webtest' (v0.1.2) from /etc/puppet/modules
[root@servera ~]# puppet module install \
      ~student/labwork/rht-webtest/pkg/rht-webtest-0.1.3.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── rht-webtest (v0.1.3)
```

8. Apply the installed module's smoke test manifest. Confirm that **/var/www/html/index.html** contains the content of the file provided by the Puppet module.

```
[root@servera ~]# puppet apply /etc/puppet/modules/webtest/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.18 seconds
... Output omitted ...
Notice: /Stage[main]/Webtest::Files/File[/var/www/html/index.html]/ensure:
 defined content as '{md5}7e69ac6558d1991aedd02348cbee33b5'
Notice: Finished catalog run in 0.74 seconds
[root@servera ~]# cat /var/www/html/index.html
<h1>Puppet provided index.html</h1>
```

```
<p>
  This is a file provided in the files/ directory.
</p>
```

9. When Puppet needs to customize the file content based on facts about the host, a template must be used instead of a fixed content file.

   Log back in as **student** on **servera**. Define a function, called **validate_erb**, that checks the syntax of a file in Puppet ERB, the templating language of Puppet. Define it in the **~/.bashrc** file to make it persist.

```
[student@servera labwork]$ cat ~/.bashrc
# .bashrc

... Output omitted ...

# User specific aliases and functions

validate_erb() {
  erb -P -x -T - $1 | ruby -c
}
[student@servera labwork]$ source ~/.bashrc
```

10. Convert the fixed format **index.html** file into a template that Puppet customizes with a fact about the system.

    10.1. Create a **templates** directory and copy the fixed content **index.html** into the directory. Add a **.erb** extension to tell Puppet that this will be a template written in the Puppet ERB language.

```
[student@servera labwork]$ cd rht-webtest
[student@servera rht-webtest]$ mkdir templates
[student@servera rht-webtest]$ cp files/index.html templates/index.html.erb
```

    10.2. Modify the **index.html.erb** template so that it contains the following content:

```
<h1>Puppet provided index.html</h1>
<p>
  This is a template provided in the templates/ directory.
</p>
<p>
  It is being hosted on <%= @fqdn %>.
</p>
```

    The **<%= @fqdn %>** expression will be replaced with the **fqdn** system fact.

    10.3. Check the syntax of the template with the **validate_erb** function.

```
 [student@servera rht-webtest]$ validate_erb templates/index.html.erb
Syntax OK
```

11. Modify the **files.pp** manifest so the template is used instead of the fixed content file in **files**. The resulting file should look like the following.

```
# == Class: webtest::files
#

class webtest::files {

  file { '/var/www/html/index.html':
    ensure  => 'file',
    mode    => '0644',
    content => template('webtest/index.html.erb'),
    require => Package['httpd'],
  }

}
```

Change the **source** parameter back to **content** and call the **template()** function.

12. Increment the module version

```
[student@servera rht-webtest]$ grep version metadata.json
  "version": "0.1.4",
```

13. Add the **templates** directory and the manifest and **metadata.json** changes to the Git staging area. Commit the changes to the local Git repository with a descriptive log message.

```
[student@servera rht-webtest]$ git add templates manifests/files.pp metadata.json
[student@servera rht-webtest]$ git commit -m 'Put index.html content in a template.'
... Output omitted ...
```

14. Build a new version of the module.

```
[student@servera rht-webtest]$ cd ..
[student@servera labwork]$ puppet module build rht-webtest
Notice: Building /root/rht-webtest for release
Module built: /root/rht-webtest/pkg/rht-webtest-0.1.4.tar.gz
```

15. Log back in as **root** on **servera** and install the newest version of the module that was just built.

```
[root@servera ~]# puppet module install --force \
      ~student/labwork/rht-webtest/pkg/rht-webtest-0.1.4.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── rht-webtest (v0.1.4)
```

16. Apply the installed module's smoke test manifest. Confirm that **/var/www/html/index.html** contains the content of the template with the fully qualified host name of **servera** substituted for the variable reference.

```
[root@servera ~]# puppet apply /etc/puppet/modules/webtest/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.06 seconds
```

```
... Output omitted ...
Notice: /Stage[main]/Webtest::Files/File[/var/www/html/index.html]/content:
 content changed '{md5}7e69ac6558d1991aedd02348cbee33b5' to
 '{md5}6e9ca937d95e30d6a4b04a2053860cf6'
Notice: Finished catalog run in 0.43 seconds
[root@servera ~]# cat /var/www/html/index.html
<h1>Puppet provided index.html</h1>
<p>
  This is a template provided in the templates/ directory.
</p>
<p>
  It is being hosted on servera.lab.example.com.
</p>
```

17. As **student**, commit any additional changes you made into the local Git repository. Once you have successfully finished the exercise, push your committed changes to the upstream Git repository.

```
[student@servera labwork]$ git push
... Output omitted ...
```

Lab: Implementing Relationships in a Puppet Module

In this lab, you will write a Puppet module, called **rht-ftpcustom**, that installs and configures a customized FTP server. The module should be written so that the service is deployed in the following order: software installation, configuration, then the service is enabled and started.

Any configuration corrections made by Puppet should cause the service to be refreshed. The FTP server must use the provided configuration and login banner files.

Resources	
Files	**/var/tmp/custom-banner** and **/var/tmp/vsftpd.conf**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet module that deploys a service using existing, intact configuration files. The module should define relationships between Puppet resources/classes so that the components of the service are deployed in a specific order.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script. It copies the provided configuration file and login banner to the **/var/tmp** directory on **servera**.

```
[student@workstation ~]$ lab puppet-relationships setup
```

2. Log in as **root** on **servera** and configure the firewall on **servera** to allow incoming FTP connections.

3. Log in as **student** on **servera** and build a new directory hierarchy for a Puppet module called **rht-ftpcustom**. Create it in the **labwork** directory so changes can be managed by Git.

4. Make sure the provided configuration and login banner files are packaged in the Puppet module.

5. Write the Puppet classes that install the *vsftpd* package, deploy provided files, and manage the **vsftpd** service. Puppet relationships should be defined so that the resources are established in the following order: software installation, configuration, then the service is enabled and started. Any corrections to the service configuration should cause the service to be refreshed.

6. Define a main smoke test manifest that will deploy the FTP service.

7. Update the module metadata so that it has no dependencies on other modules.

8. Add the module source directory to the Git staging area. Commit the changes with a descriptive log message.

9. Build and install the **rht-ftpcustom** module.

10. Test the FTP service to make sure it is configured properly. The following output displays when a successful connection is made:

```
[student@workstation ~]$ ftp servera.lab.example.com
Connected to servera.lab.example.com (172.25.250.10).
220-############################################################
220-#################### Private FTP Server ####################
220-############# Only authorized users permitted. #############
220-############################################################
220
Name (servera.lab.example.com:student):
```

11. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-relationships grade
```

12. As **student** on **servera**, commit any additional changes into the local Git repository. Once you have successfully finished the lab, push your recent changes to the upstream Git repository.

Solution

In this lab, you will write a Puppet module, called **rht-ftpcustom**, that installs and configures a customized FTP server. The module should be written so that the service is deployed in the following order: software installation, configuration, then the service is enabled and started.

Any configuration corrections made by Puppet should cause the service to be refreshed. The FTP server must use the provided configuration and login banner files.

Resources	
Files	**/var/tmp/custom-banner** and **/var/tmp/vsftpd.conf**
Machines	**servera**

Outcome(s)
You should be able to write a Puppet module that deploys a service using existing, intact configuration files. The module should define relationships between Puppet resources/classes so that the components of the service are deployed in a specific order.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** as **student** and run the lab setup script. It copies the provided configuration file and login banner to the **/var/tmp** directory on **servera**.

```
[student@workstation ~]$ lab puppet-relationships setup
```

2. Log in as **root** on **servera** and configure the firewall on **servera** to allow incoming FTP connections.

```
[root@servera ~]# firewall-cmd --add-service=ftp
success
[root@servera ~]# firewall-cmd --permanent --add-service=ftp
success
```

3. Log in as **student** on **servera** and build a new directory hierarchy for a Puppet module called **rht-ftpcustom**. Create it in the **labwork** directory so changes can be managed by Git.

   Use the **puppet module generate** command to create the directory hierarchy and Puppet module metadata.

```
[student@servera ~]$ cd labwork
[student@servera labwork]$ puppet module generate --skip-interview rht-ftpcustom
```

4. Make sure the provided configuration and login banner files are packaged in the Puppet module.

   Make a **files** directory in the module development directory and copy the provided configuration files there. Templates are not needed because the files do not have to be modified in any way.

```
[student@servera labwork]$ mkdir -p rht-ftpcustom/files
[student@servera labwork]$ cp /var/tmp/{custom-banner,vsftpd.conf} rht-ftpcustom/
files
```

5. Write the Puppet classes that install the *vsftpd* package, deploy provided files, and manage the **vsftpd** service. Puppet relationships should be defined so that the resources are established in the following order: software installation, configuration, then the service is enabled and started. Any corrections to the service configuration should cause the service to be refreshed.

   5.1. Create a Puppet class that will install the necessary *vsftpd* package. Save it to a file that is named after the Puppet class, in this case **manifests/install.pp**.

```
class ftpcustom::install {

  package { 'vsftpd':
    ensure  => 'present',
  }

}
```

   5.2. Create a Puppet class that will configure the FTP service by installing the provided **custom-banner** and **vsftpd.conf** files. Save it to a file that is named after the Puppet class, in this case **manifests/config.pp**. Using the **source** parameter in the **file** resources will ensure the file contents are left unchanged.

```
class ftpcustom::config {

  file { '/etc/vsftpd/vsftpd.conf':
    ensure   => 'file',
    mode     => '600',
    source   => 'puppet:///modules/ftpcustom/vsftpd.conf',
  }

  file { '/etc/vsftpd/custom-banner':
    ensure   => 'file',
    mode     => '644',
    source   => 'puppet:///modules/ftpcustom/custom-banner',
  }

}
```

   5.3. Create a Puppet class that will start and enable the **vsftpd** service. Save it to a file that is named after the Puppet class, in this case **manifests/service.pp**.

```
class ftpcustom::service {

  service { 'vsftpd':
    ensure  => 'running',
    enable  => true,
    require => Package['vsftpd'],
  }

}
```

5.4. Modify the **ftpcustom** Puppet class that is defined in **manifests/init.pp**. It should include the other Puppet classes that are defined for this module. Dependencies could have been defined with the **before**, **require**, **notify**, and **subscribe** metaparameters; the following uses a timeline instead.

```
[student@servera labwork]$ vim rht-ftpcustom/manifests/init.pp
[student@servera labwork]$ tail rht-ftpcustom/manifests/init.pp
#
class ftpcustom {

  class { '::ftpcustom::install': } ->
  class { '::ftpcustom::config': } ~>
  class { '::ftpcustom::service': }

}
```

6. Define a main smoke test manifest that will deploy the FTP service.

   The **puppet module generate** command will generate an appropriate smoke test by default. If the Puppet module is created manually, make sure the smoke test manifest includes the **ftpcustom** class.

```
# The baseline for module testing used by Puppet Labs is that each manifest
# should have a corresponding test manifest that declares that class or defined
# type.
#
# Tests are then run by using puppet apply --noop (to check for compilation
# errors and view a log of events) or by fully applying the test in a virtual
# environment (to compare the resulting system state to the desired state).
#
# Learn more about module testing here:
# http://docs.puppetlabs.com/guides/tests_smoke.html
#
include ftpcustom
```

7. Update the module metadata so that it has no dependencies on other modules.

   Update the **dependencies** field in the **metadata.json** file.

```
{
  "name": "rht-ftpcustom",
  "version": "0.1.0",
  "author": "rht",
  "summary": null,
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
  "issues_url": null,
  "dependencies": [
  ]
}
```

8. Add the module source directory to the Git staging area. Commit the changes with a descriptive log message.

```
[student@servera labwork]$ git add rht-ftpcustom
[student@servera labwork]$ git commit -m 'Initial version of rht-ftpcustom.'
... Output omitted ...
```

9. Build and install the **rht-ftpcustom** module.

   9.1. The module can be built by an unprivileged user.

```
[student@servera labwork]$ puppet module build rht-ftpcustom
Notice: Building /home/student/labwork/rht-ftpcustom for release
Module built: /home/student/labwork/rht-ftpcustom/pkg/rht-ftpcustom-0.1.0.tar.gz
```

   9.2. **root** privileges are required to install the module.

```
[root@servera ~]# puppet module install ~student/labwork/rht-ftpcustom/pkg/rht-
ftpcustom-0.1.0.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── rht-ftpcustom (v0.1.0)
```

   9.3. **root** privileges are also required to smoke-test the module.

```
[root@servera ~]# puppet apply /etc/puppet/modules/ftpcustom/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.96 seconds
... Output omitted ...
Notice: /Stage[main]/Ftpcustom::Install/Package[vsftpd]/ensure: created
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/content:
 content changed '{md5}c4072ca90053a6e86cf86850c343346d' to
 '{md5}81ee930dab4087a96ea4f55b699702d2'
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/custom-banner]/ensure:
 defined content as '{md5}61c72b9a1e7bf54059d3b680dd1a0929'
Notice: /Stage[main]/Ftpcustom::Service/Service[vsftpd]/ensure: ensure
 changed 'stopped' to 'running'
Notice: Finished catalog run in 7.41 seconds
```

10. Test the FTP service to make sure it is configured properly. The following output displays when a successful connection is made:

```
[student@workstation ~]$ ftp servera.lab.example.com
Connected to servera.lab.example.com (172.25.250.10).
220-########################################################################
220-#################### Private FTP Server ####################
220-############# Only authorized users permitted. #############
220-########################################################################
220
Name (servera.lab.example.com:student):
```

It would also be a good idea to test the relationship between the configuration files and the service. If Puppet restores a configuration file, the service should be refreshed.

```
[root@servera ~]# echo > /etc/vsftpd/vsftpd.conf
```

```
[root@servera ~]# puppet apply /etc/puppet/modules/ftpcustom/tests/init.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 1.01 seconds
... Output omitted ...
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/content:
 content changed '{md5}68b329da9893e34099c7d8ad5cb9c940' to
'{md5}81ee930dab4087a96ea4f55b699702d2'
Notice: /Stage[main]/Ftpcustom::Service/Service[vsftpd]: Triggered
 'refresh' from 1 events
Notice: Finished catalog run in 0.68 seconds
```

11. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-relationships grade
```

12. As **student** on **servera**, commit any additional changes into the local Git repository. Once you have successfully finished the lab, push your recent changes to the upstream Git repository.

```
[student@servera labwork]$ git push
... Output omitted ...
```

Summary

In this chapter, you learned:

- Namespaces define the file path of the files in a module that the Puppet autoloader searches for the definition of a Puppet object. The first element is the module to be searched and the remaining elements are the directories to search for the object.
- The **`require`** and **`before`** metaparameters specify an ordered relationship between Puppet resources.
- The **`subscribe`** and **`notify`** metaparameters also specify an ordered relationship between resources, but the dependent resource is refreshed when its dependency changes state.
- References to Puppet objects start with an uppercase letter and square braces enclose the title of the resource being referenced.
- Puppet modules can publish static files in the **`files`** directory for use by their resources. They are referenced in **`file`** resources with the **`source`** parameter.
- Dynamic file content can be published in the **`templates`** directory of a module. They are included in the **`content`** parameters of **`file`** resources by calling the **`template()`** function.

CHAPTER 9

IMPLEMENTING VARIABLES AND CONDITIONALS IN A PUPPET MODULE

Overview	
Goal	Implement variables and conditionals in a Puppet module.
Objectives	• Implement variables in a Puppet module. • Implement conditionals in a Puppet module. • Implement regular expressions in a Puppet module.
Sections	• Implementing Variables (and Guided Exercise) • Implementing Conditionals (and Guided Exercise) • Implementing Regular Expressions (and Guided Exercise)
Lab	• Implementing Variables and Conditionals in a Puppet Module

Implementing Variables

Objectives

After completing this section, students should be able to:

- Implement variables in a Puppet module.

Using variables

As in other scripting languages, Puppet variables provide storage for values that can be retrieved for later use. In Puppet DSL, variable names are denoted with a **$** prefix. They are case-sensitive and can be composed of alphanumeric characters and underscores. Variable values are assigned using the **=** operator.

The Puppet DSL provides several different data types, including Booleans, strings, numbers, arrays, hashes, and regular expressions. With the exception of regular expressions, variables can be assigned value of any data type.

The following are examples of variable assignment involving string, number, array, and hash.

```
$boolean_var = true
$number_var = 5
$string_var = "Custom message.\n"
$array_var = [ 'a', 'b', 'c' ]
$hash_var = {
 'first'  => 'primary'
 'second' => 'secondary'
 'third'  => 'tertiary'
}
```

After a variable has been assigned a value, the value can be retrieved by the variable name. Puppet will perform variable substitution and replace the variable with its value. When variables are enclosed within double quotes, Puppet will replace them with their values. However, variables enclosed within single quotes are treated as literals and therefore will not be interpreted.

When retrieving values from variables, some contexts require additional clarification. For example, a variable name may be ambiguous due to the absence of white-space characters separating it from trailing characters. In these situations, the variable should be enclosed within **{}** to clearly delineate the boundaries of the variable name.

```
my $foo = 'name'
my $foobar = 'birthday'

notify { "My $foobar": }
notice { "My ${foo}bar": }
```

In programming, the term variable refers to a storage whose content can change or is variable. The term constant refers to a storage whose content does not change. In Puppet, the term variable is slightly misleading. Once a Puppet variable is assigned a value, it cannot be reassigned a value within the same scope that the original variable assignment occurred in.

Understanding variable scope

As in other programming languages, a scope refers to the portion of code in a program within which a variable is valid. A variable is valid within the scope it is defined, as well as within any child scope. Therefore, while the contents of a variable defined in a parent scope can be accessed from within a child scope, the reverse is not true. A variable defined within a child scope cannot have its contents retrieved from within its parent scope.

In Puppet, the top scope is the parent of all scopes. Any variables defined in this scope can be accessed by all scopes since all scopes are children of the top scope.

The next level of scoping inside of the top scope is the node scope. The node scope is created by node definitions, which will be covered in greater detail later in this course. Since each node can only match one node definition, there can only be one node scope.

```
# site.pp

$top_var = 'TOP'

node 'sandbox.example.com' {
  $node_var = "NODE"
  notify {"Node view of node: $node_var":}
  notify {"Node view of top: $top_var":}
}

notify {"Top view of node: $node_var":}
```

```
[student@sandbox] $ puppet apply site.pp
notice: Node view of node: NODE
notice: Node view of top: TOP
notice: Top view of node:
```

Within the node scope is one or more class scopes, which correlate to each Puppet class defined. Within each class scope, one can access the variables from the top scope and the node scope. However, one is not able to access variables defined in other class scopes.

While a variable can only be assigned once within a scope, this limitation does not apply across scopes. A variable inherited by a scope from a parent scope can be reassigned a new value.

Built-in variables

The previous examples revolved around custom defined variables. In addition to custom defined variables, Puppet also offers some predefined variables. Termed built-in variables, these variables are created by **`facter`** and populated with the facts it discovers.

These variables are defined in the top scope and therefore is accessible from anywhere within a Puppet manifest. These built-in variables include not just the ones which come included by default with Puppet, also called core facts, but also custom facts defined by users.

Built-in variables are named no differently than other variables. This makes it difficult to distinguish them from user-defined variables. Also, since built-in variables are defined in top scope, it is possible for them to be overridden in a child scope if a variable of the same name is defined within the child scope. One way to avoid these issues is to use a fully qualified syntax for the built-in variable name. Using the variable name **`$::`*`FACTNAME`***, instead of **`$`*`FACTNAME`***, allows built-in variables to be specifically referred to from any scope and allows them to be distinguished from user-defined variables.

References

Puppet Variables
https://docs.puppetlabs.com/puppet/3.6/reference/lang_variables.html

Puppet Scope
https://docs.puppetlabs.com/puppet/3.6/reference/lang_scope.html

Puppet Facts and Built-in Variables
https://docs.puppetlabs.com/puppet/3.6/reference/lang_facts_and_builtin_vars.html

Facter 3.1: Custom Facts Walkthrough
https://docs.puppetlabs.com/facter/latest/custom_facts.html

Guided Exercise: Implementing Variables

In this lab, you will learn to define and use variables in a Puppet manifest.

Resources	
Machines	**servera**

Outcome(s)
You should be able to define and use a Puppet variable in a manifest.

Before you begin
The **servera** host should already have Puppet and Git installed.

1. Log in as **root** on **servera**. Change into the **labwork** directory and pull any upstream changes into the local Git repository and working tree.

```
[root@servera ~]# cd labwork
[root@servera labwork]# git pull
... Output omitted ...
```

2. Start by defining a variable and using it once in a Puppet manifest.

   2.1. Create a Puppet manifest called **vars.pp**. It should define a variable, **$dir**, that is a path to a directory that needs to exist on the system. A **file** resource should be defined that uses the **$dir** variable. The manifest should look something like the following:

```
$dir = '/var/tmp/example'

file { "$dir":
  ensure => 'directory',
  mode   => '755',
}
```

   2.2. Apply the **vars.pp** manifest. The new directory that is created is named according to the value of the **$dir** variable.

```
[root@servera labwork]# puppet apply vars.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.19 seconds
Notice: /Stage[main]/Main/File[/var/tmp/example]/ensure: created
Notice: Finished catalog run in 0.05 seconds
[root@servera labwork]# ls -ld /var/tmp/example
drwxr-xr-x. 2 root root 6 Sep  8 15:23 /var/tmp/example
```

   2.3. Commit **vars.pp** to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add vars.pp
[root@servera labwork]# git commit -m 'Initial version.'
... Output omitted ...
```

3. Puppet variables are valuable because they can be defined once, then be used multiple times for related objects within the same manifest.

   3.1. Add a couple more **file** resources to **vars.pp** to manage a world-readable file called **/var/tmp/example/public.txt** and a private file called **/var/tmp/example/secret.txt**. Be sure to use the **$dir** variable where possible. The resulting manifest should look something like the following:

```
... Output omitted ...

file { "$dir/public.txt":
  ensure  => 'file',
  mode    => '644',
  content => "Public content\n",
}

file { "$dir/secret.txt":
  ensure  => 'file',
  mode    => '600',
  content => "Secret content\n",
}
```

   3.2. Apply the Puppet manifest and see the adjustments it makes to the file system. Verify the contents of the new files.

```
[root@servera labwork]# puppet apply vars.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.20 seconds
Notice: /Stage[main]/Main/File[/var/tmp/example/public.txt]/ensure: defined
 content as '{md5}5ebb85561c44250e747fc25bda1f94cd'
Notice: /Stage[main]/Main/File[/var/tmp/example/secret.txt]/ensure: defined
 content as '{md5}9943fb5ded3b0a575b6c33e7ce304b9d'
Notice: Finished catalog run in 0.08 seconds
[root@servera labwork]# ls -l /var/tmp/example
total 8
-rw-r--r--. 1 root root 15 Sep  8 15:24 public.txt
-rw-------. 1 root root 15 Sep  8 15:24 secret.txt
[root@servera labwork]# more /var/tmp/example/*
::::::::::::::
/var/tmp/example/public.txt
::::::::::::::
Public content
::::::::::::::
/var/tmp/example/secret.txt
::::::::::::::
Secret content
```

   3.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add vars.pp
[root@servera labwork]# git commit -m 'Added file resources that use variables.'
... Output omitted ...
```

4. Puppet variables can only be assigned a value once.

   4.1. Add another line to **vars.pp** that tries to change the value of **$dir**, just before the last **file** resource. The resulting manifest should look like the following:

```
... Output omitted ...

file { "$dir/public.txt":
  ensure  => 'file',
  mode    => '644',
  content => "Public content\n",
}

$dir = '/tmp/example'

file { "$dir/secret.txt":
  ensure  => 'file',
  mode    => '600',
  content => "Secret content\n",
}
```

4.2. Try to apply the updated manifest.

```
[root@servera labwork]# puppet apply vars.pp
Error: Cannot reassign variable dir at /root/labwork/vars.pp:14 on node
 servera.lab.example.com
Error: Cannot reassign variable dir at /root/labwork/vars.pp:14 on node
 servera.lab.example.com
```

The **puppet apply** command fails because of the second variable definition.

4.3. Use Git to discard the changes and restore the manifest to its original working condition.

```
[root@servera labwork]# git checkout -- vars.pp
```

5. System facts provided by Facter are variables that can be referenced.

5.1. Modify the Puppet manifest so that it puts the value of the **osfamily** system fact in the private file, **secret.txt**. The string should be enclosed with double quotes so the variable reference will be recognized.

```
... Output omitted ...

file { "$dir/secret.txt":
  ensure  => 'file',
  mode    => '600',
  content => "The osfamily for this system is $::osfamily.\n",
}
```

5.2. Use the **facter** command to display the value of the **osfamily** system fact.

```
[root@servera labwork]# facter osfamily
RedHat
```

5.3. Apply the manifest and display the new contents of the **/var/tmp/example/secret.txt** file.

```
[root@servera labwork]# puppet apply vars.pp
```

```
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.20 seconds
Notice: /Stage[main]/Main/File[/var/tmp/example/secret.txt]/content:
 content changed '{md5}9943fb5ded3b0a575b6c33e7ce304b9d' to
 '{md5}55636325d2aa3adc5644bb1a4785ec82'
Notice: Finished catalog run in 0.10 seconds
[root@servera labwork]# cat /var/tmp/example/secret.txt
The osfamily for this system is RedHat.
```

6. When a variable is immediately surrounded by text, its name must be delimited with braces (**{ }**).

   6.1. Change the Puppet manifest so the content for the **public.txt** file uses the **$dir** variable in a continuous string of text. Be sure to enclose the variable name in braces.

```
... Output omitted ...

file { "$dir/public.txt":
  ensure  => 'file',
  mode    => '644',
  content => "Here is a variable${dir}immediately enclosed in text.\n",
}

... Output omitted ...
```

   6.2. Apply the manifest and observe the changes it makes to **/var/tmp/example/public.txt**.

```
[root@servera labwork]# puppet apply vars.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.19 seconds
Notice: /Stage[main]/Main/File[/var/tmp/example/public.txt]/content:
 content changed '{md5}5ebb85561c44250e747fc25bda1f94cd' to
 '{md5}bce844cb785b7ddb157e180bdb933515'
Notice: Finished catalog run in 0.09 seconds
[root@servera labwork]# cat /var/tmp/example/public.txt
Here is a variable/var/tmp/exampleimmediately enclosed in text.
```

   6.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add vars.pp
[root@servera labwork]# git commit -m 'Use facts and {}s.'
... Output omitted ...
```

7. Push your committed changes to the upstream Git repository.

```
[root@servera labwork]# git push
... Output omitted ...
```

Implementing Conditionals

Objectives

After completing this section, students should be able to:

- Implement conditionals in a Puppet module.

Using conditionals

While Puppet manifests declare the desired end states of systems, they do not dictate how those states will be achieved. The reason for this is that in order for Puppet to be a cross-platform tool, it needs to perform different actions on different systems in order to achieve the same outcome. For example, while it is possible to run a network time server on both RHEL 6 and RHEL 7 systems, the former is accomplished using the **`ntpd`** service, while the latter can be accomplished with the use of the **`chronyd`** service.

In a large enterprise environment, there will likely be a variety of systems that have their configurations managed by Puppet. Differences between systems may require slightly different Puppet code for the same configuration objective. While it is possible to implement different sets of code for different systems, it is much more desirable to implement decision-making logic into the code to accommodate for the differences. Doing so allows for code reuse and eases the management of Puppet code as an organization's Puppet infrastructure grows.

Like many programming languages, the Puppet DSL offers administrators the ability to add conditional statements into their code. Per its namesake, conditional statements allow for different code to be executed for different conditions. With Puppet configuration management, these conditions are typically based on system information derived from Puppet facts.

Conditional operators

Conditional expressions are created with the use of operators and operands. The most common type of operators are comparison operators. Comparison operators can evaluate strings or numeric values.

The following table lists some operators commonly used for string comparison.

Commonly Used String Comparison Operators

Operator	Description
`==`	Tests whether two strings are identical.
`!=`	Tests whether two strings are not identical.
`in`	Tests whether the right operand contains the left operand. Most commonly used with a left string operand and a right array operand to see if the string matches an element in the array.

The following table lists some operators commonly used for numeric comparison.

Commonly Used Numeric Comparison Operators

Operator	Description
`==`	Tests whether two numbers are equal.
`!=`	Tests whether two numbers are not equal.

Operator	Description
<	Tests whether the left operand is less than the right operand.
<=	Tests whether the left operand is less than or equal to the right operand.
>	Tests whether the left operand is greater than the right operand.
>=	Tests whether the left operand is greater than or equal to the right operand.

Conditional expressions can be combined together to create compound expressions with the use of the **and** and **or** operators. The **and** operator tests whether both operand conditional expressions it evaluates are true. The **or** operator tests whether either one of the operand conditional expressions it evaluates are true.

Using `if` statements

Puppet offers several types of conditional statements, each of which is suited for different scenarios. The most basic type is the classic ***`if` statement***. The simplest form of the **`if`** statement executes a code block when a condition is evaluated to be true.

```
if $::is_virtual == 'true' {
... Output omitted ...
}
```

The **`if`** statement can also be extended with the **`else`** keyword to specify one set of actions when a condition is true and another set of actions when the condition is false.

```
if $::is_virtual == 'true' {
... Output omitted ...
} else {
... Output omitted ...
}
```

Lastly, an **`elsif`** keyword can also be used to extend an **`if`** statement to test more than one condition. This structure allows for different actions to be performed for each condition and for a catchall code block to be executed if none of the conditions prove to be true.

```
if $::is_virtual == 'true' {
... Output omitted ...
} elsif {
... Output omitted ...
} else {
... Output omitted ...
}
```

Using `unless` statements

While the **`if`** statement is primarily used to perform a set of actions when a condition is true, there are often times when it is desirable to execute code in the opposite scenario when a condition is false. For these situations, Puppet provides the ***`unless` statement***.

The **`unless`** statement should be used when testing for a false condition is easier and more efficient than if an **`if`** statement were used.

The following example demonstrates a situation where the use of the **unless** statement is more programmatically efficient and understandable than it would be with the use of an **if** statement.

```
unless $memorysize >= 2048 {
... Output omitted ...
}
```

Using case statements

While the **if** statement can be extended with one or more optional **elsif** keywords to allow different code execution for different conditions, excessive use of the **elsif** keywords can make code readability difficult. Puppet offers the **case** statement as an alternative conditional statement in these situations.

With a **case** statement, a list of values are evaluated against the control expression. When a value evaluates true against a control expression, its corresponding code block is executed and the **case** statement is then exited.

The **else** keyword in an **if** statement allows for the execution of a code block if none of the conditions evaluated true. Similarly, the **case** statement offers a **default** value that serves as a catchall if no previous values evaluated successfully against a control expression.

```
case $operatingsystem {
  /Linux/:     { notify 'Powered by Linux.': }
  'Solaris':   { notify 'Powered by Solaris.': }
  default:     { notify 'Welcome.': }
}
```

Using selector statements

Another type of conditional statement offered by Puppet is the *selector statement*. Like a **case** statement, a list of values are evaluated against a control expression in a selector statement. However, when a value evaluates true against a control expression, rather than executing a block of code, a value is returned instead.

The selector statement is composed of several components: a control variable, a case, and a value. The control variable is evaluated against each case. When a case evaluates true against the control variable, its corresponding value is returned. Similar to the **case** statement, the selector statement also offers a **default** case that serves as a catchall if no previous case evaluated successfully against the control variable.

```
$control_variable ? {
  'case_1' => 'return_value_1',
  'case_2' => 'return_value_2',
  default  => 'default_return_value',
}
```

Due to its nature, a selector statement is most useful for situations where variable assignment logic varies depending on certain condition. The following example demonstrates the use of a selector statement for variable assignment.

```
$message = $operatingsystem ? {
  /Linux/      => 'Powered by Linux.',
  'Solaris'    => 'Powered by Solaris.',
```

```
  default      => 'Welcome.',
}
```

References

Puppet Conditional Statements
http://docs.puppetlabs.com/puppet/3.6/reference/lang_conditional.html

Guided Exercise: Implementing Conditionals

In this lab, you will learn how to use conditionals in a Puppet manifest.

Resources	
Machines	**servera**

Outcome(s)
You should be able to use selectors to return different values based on a condition. You should also be able to use the **if**, **if-else**, **if-elsif-else**, **unless**, and **case** conditionals in a Puppet manifest.

Before you begin
The **servera** host should already have Puppet and Git installed.

1. Log in as **root** on **servera**. Change into the **labwork** directory and pull any upstream changes into the local Git repository and working tree.

```
[root@servera ~]# cd labwork
[root@servera labwork]# git pull
... Output omitted ...
```

2. Start by observing how a selector can be used to return a variety of values based on the value of an expression.

   2.1. Create a manifest called **condition.pp** that sets the value of a variable, named **$disk**, based on a selector. The selector will use the **is_virtual** system fact to decide which value to return.

   Use a **notify** resource to display the final value of **$disk**. The resulting manifest should look like the following:

```
$disk = $::is_virtual ? {
  'true'  => '/dev/vda',
  default => '/dev/sda',
}

notify { 'disk':
  message => "value is $disk.",
}
```

   2.2. Use the **facter** command to display the value of the **is_virtual** fact.

```
[root@servera labwork]# facter is_virtual
true
```

   2.3. Apply the manifest and confirm that the selector returns **/dev/vda**, so that is the value that is assigned to the **$disk** variable.

```
[root@servera ~]# puppet apply condition.pp
Notice: Compiled catalog for servera.lab.example.com in environment
```

```
 production in 0.04 seconds
Notice: value is /dev/vda.
Notice: /Stage[main]/Main/Notify[disk]/message: defined 'message' as
 'value is /dev/vda.'
Notice: Finished catalog run in 0.05 seconds
```

2.4. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add condition.pp
[root@servera labwork]# git commit -m 'Initial version.'
... Output omitted ...
```

3. Use an **if** conditional to optionally apply a **file** resource in a manifest.

3.1. Extend **condition.pp** so that it looks like the following:

```
$disk = $::is_virtual ? {
  'true'  => '/dev/vda',
  default => '/dev/sda',
}

notify { 'disk':
  message => "value is $disk.",
}

if $::is_virtual {
  file { '/tmp/virtual.txt':
    ensure  => 'file',
    content => "This is a virtual system. Disk is ${disk}.\n",
  }
}
```

3.2. Use the **puppet apply** command to apply the manifest. Since the **is_virtual** fact evaluates to **true**, the resource defined in the **if** condition is applied and the file is created.

```
[root@servera ~]# puppet apply condition.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.20 seconds
Notice: /Stage[main]/Main/File[/tmp/virtual.txt]/ensure: defined content
 as '{md5}815b89d5a2775cce3d5e9ba12e832544'
Notice: value is /dev/vda.
Notice: /Stage[main]/Main/Notify[disk]/message: defined 'message' as
 'value is /dev/vda.'
Notice: Finished catalog run in 0.05 seconds
[root@servera ~]# cat /tmp/virtual.txt
This is a virtual system. Disk is /dev/vda.
```

3.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add condition.pp
[root@servera labwork]# git commit -m 'Added an if statement.'
... Output omitted ...
```

4. Reverse the logic of the **if** conditional by changing it to an **unless** conditional.

4.1. Modify **condition.pp** to look like the following:

```
$disk = $::is_virtual ? {
  'true'  => '/dev/vda',
  default => '/dev/sda',
}

notify { 'disk':
  message => "value is $disk.",
}

unless $::is_virtual {
  file { '/tmp/physical.txt':
    ensure  => 'file',
    content => "This is a physical system. Disk is ${disk}.\n",
  }
}
```

Note the name of the file and that its content was changed to reflect the reverse nature of the logic.

4.2. Apply the Puppet manifest. The **/tmp/physical.txt** file does not get created because the expression used with the **unless** condition is **true**, so the associated code is skipped.

```
[root@servera ~]# puppet apply condition.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: value is /dev/vda.
Notice: /Stage[main]/Main/Notify[disk]/message: defined 'message' as
 'value is /dev/vda.'
Notice: Finished catalog run in 0.05 seconds
[root@servera ~]# ls /tmp/physical.txt
ls: cannot access /tmp/physical.txt: No such file or directory
```

5. Observe how the **if** conditional works with an **else** clause.

5.1. Modify the **condition.pp** manifest. Change the **unless** conditional back to an **if** conditional and add an **else** clause so the manifest looks like the following:

```
$disk = $::is_virtual ? {
  'true'  => '/dev/vda',
  default => '/dev/sda',
}

notify { 'disk':
  message => "value is $disk.",
}

if $::is_virtual {
  file { '/tmp/virtual.txt':
    ensure  => 'file',
    content => "This is a virtual system. Disk is ${disk}.\n",
  }
}
else {
  file { '/tmp/physical.txt':
    ensure  => 'file',
```

```
    content => "This is a physical system. Disk is ${disk}.\n",
  }
}
```

5.2. Clean up from previous Puppet actions by removing all the files in **/tmp** that end with **.txt**. Apply the manifest and confirm **virtual.txt** is created.

```
[root@servera ~]# rm -f /tmp/*.txt
[root@servera ~]# puppet apply condition.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.20 seconds
Notice: /Stage[main]/Main/File[/tmp/virtual.txt]/ensure: defined content
 as '{md5}815b89d5a2775cce3d5e9ba12e832544'
Notice: value is /dev/vda.
Notice: /Stage[main]/Main/Notify[disk]/message: defined 'message' as
 'value is /dev/vda.'
Notice: Finished catalog run in 0.05 seconds
```

Since the condition was **true**, the first clause was applied and the **else** clause was skipped.

5.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add condition.pp
[root@servera labwork]# git commit -m 'Changed if to if/else.'
... Output omitted ...
```

6. Implement an **if-elsif-else** condition that branches based on the value of the **osfamily** system fact.

6.1. Create a manifest called **os-check.pp** that implements the following Puppet DSL:

```
if $::osfamily == 'RedHat' or $::osfamily == 'CentOS' {
  notify { "This is a Red Hat family system." : }
}
elsif $::osfamily == 'Solaris' {
  notify { "This is a Solaris system." : }
}
else {
  notify { "Not sure what OS family this system is." : }
}
```

6.2. Use the **facter** command to display the value of the **osfamily** fact, then apply the manifest. Since the first condition evaluates to **true**, it implements the **notify** resource and skips the remaining clauses of that condition.

```
[root@servera ~]# facter osfamily
RedHat
[root@servera ~]# puppet apply os-check.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: This is a Red Hat family system.
Notice: /Stage[main]/Main/Notify[This is a Red Hat family system.]/
 message: defined 'message' as 'This is a Red Hat family system.'
Notice: Finished catalog run in 0.05 seconds
```

6.3. Commit the new file to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add os-check.pp
[root@servera labwork]# git commit -m 'Initial version.'
... Output omitted ...
```

7. Implement the previous conditional using a **case** statement instead of an **if-elsif-else** statement.

7.1. Modify the **os-check.pp** manifest so it looks like the following:

```
case $::osfamily {
  'RedHat', 'CentOS': {
    notify { "This is a Red Hat family system." : }
  }
  'Solaris': {
    notify { "This is a Solaris system." : }
  }
  default: {
    notify { "Not sure what OS family this system is." : }
  }
}
```

7.2. When you apply the manifest, it will display the same notification because the logic of the **case** conditional is the same as the original code.

```
[root@servera ~]# puppet apply os-check.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: This is a Red Hat family system.
Notice: /Stage[main]/Main/Notify[This is a Red Hat family system.]/
 message: defined 'message' as 'This is a Red Hat family system.'
Notice: Finished catalog run in 0.05 seconds
```

7.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add os-check.pp
[root@servera labwork]# git commit -m 'Changed if logic to case.'
... Output omitted ...
```

8. Push your committed changes to the upstream Git repository.

```
[root@servera labwork]# git push
... Output omitted ...
```

Implementing Regular Expressions

Objectives

After completing this section, students should be able to:

- Implement regular expressions in a Puppet module.

Using regular expressions

Regular expressions, also commonly referred to by its abbreviated form, *regexp*, is a sequence of characters used to define a search pattern. In programming, regular expressions are primarily used for pattern matching of strings.

The Puppet DSL provides regular expression capabilities. These capabilities allow users to implement conditional expressions for string comparison, which are more powerful and flexible than the basic equality evaluation.

Regular expression-matching operators

Puppet offers two operators for use with regular expressions. The **=~** operator is used to evaluate whether a string matches a regular expression. The **!~** operator performs the exact opposite and evaluates whether a string does not match a regular expression.

Both the **=~** and **!~** regular expression-matching operators evaluate a string against a regular expression pattern. Puppet regular expressions are enclosed by the **/** characters. The syntax for this usage follows the format where the string being evaluated is positioned to the left of the operator and the regular expression used is positioned to the right of the operator.

```
if $hostname =~ /myhost/ {
  notify { 'Matched myhost' : }
}
```

```
if $hostname !~ /myhost/ {
  notify { 'Did not match myhost' : }
}
```

When the **=~** operator is used, the string comparison will evaluate to true if the string matches the regular expression and vice versa. When the **!~** operator is used, the string comparison will evaluate to false if the string matches the regular expression and vice versa.

The regular expression-matching operators are commonly used in the conditional expressions of **if** statements. However, they can also be used for the evaluation of cases in selectors and **case** statements.

```
$message = $operatingsystem ? {
  /Linux/      => 'Powered by Linux.',
  'Solaris'    => 'Powered by Solaris.',
  default      => 'Welcome.',
}
```

```
case $operatingsystem {
  /Linux/:     { notify 'Powered by Linux.': }
  'Solaris':   { notify 'Powered by Solaris.': }
```

```
  default:     { notify 'Welcome.': }
}
```

Regular expression syntax

Puppet utilizes the standard Ruby regular expressions. Therefore, the regular expressions valid for a Puppet installation are determined by the version of Ruby that accompanies it.

Similar to regular expressions used in other programming languages, Puppet's regular expressions also make use of a set of metacharacters. These metacharacters have special meaning within a regular expression. Therefore, if a metacharacter is to be matched literally, it must be escaped with a backslash.

The following table shows a list of metacharacters commonly used in Puppet regular expressions for matching single characters.

Commonly Used Character Matching Metacharacters

Metacharacter	Description
.	Matches a single character.
[ ]	Matches a single character within the brackets.
[^]	Matches a single character not contained within the brackets.
\w	Matches a single alphanumeric character.
\W	Matches a single nonalphanumeric character.
\d	Matches a single digit character.
\D	Matches a single nondigit character.
\s	Matches a single space character.
\S	Matches a single nonspace character.

Often, when matching a string, it is necessary to match a series of characters. When the series consists of the same character or characters of the same character class, the regular expression can be extended with the use of repetition modifiers.

The following table shows a list of modifiers commonly used in Puppet regular expressions for denoting character repetitions.

Commonly Used Repetition Modifiers

Modifier	Description
*****	Matches the preceding element zero or more times.
?	Matches the preceding element zero or one time.
+	Matches the preceding element one or more times.
{x}	Matches the preceding element exactly *x* times.
{x,}	Matches the preceding element *x* or more times.
{,y}	Matches the preceding element *y* or less times.
{x,y}	Matches the preceding element at least *x* but not more than *y* times.

Character-matching regular expressions will match against a string if the character or characters exist anywhere within the string. At times, it is desirable for a match to evaluate true only when the regular expression matches at a specific position within the string.

The following table shows a list of metacharacters commonly used in Puppet regular expressions for positional matching.

Commonly Used Positional Metacharacters

Metacharacter	Description
^	Matches the beginning of the line.
$	Matches the end of the line.

References

For more information, see

Ruby Regular Expressions
http://ruby-doc.org/core-2.2.3/Regexp.html

Guided Exercise: Implementing Regular Expressions

In this lab, you will learn how to use regular expressions in a Puppet manifest.

Resources	
Machines	**servera**

Outcome(s)
You should be able to use regular expressions to match patterns in strings.

Before you begin
The **servera** host should already have Puppet and Git installed.

1. Log in as **root** on **servera**. Change into the **labwork** directory and pull any upstream changes into the local Git repository and working tree.

```
[root@servera ~]# cd labwork
[root@servera labwork]# git pull
... Output omitted ...
```

2. Use the **facter** command to display the value of the **ipaddress** system fact.

```
[root@servera labwork]# facter ipaddress
172.25.250.10
```

   We will use it in conditional expressions to see what regular expressions can be used to match it.

3. The **==** operator checks for exact string matches.

   3.1. Create a manifest called **regex.pp** that contains the following Puppet DSL:

```
if $::ipaddress == '172.25.250.11' {
  notify { "Matched." : }
}
```

   3.2. Apply the manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Finished catalog run in 0.05 seconds
```

   The **notify** resource was not applied because the system fact did not match the string exactly.

   3.3. Commit **regex.pp** to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add regex.pp
[root@servera labwork]# git commit -m 'Initial version.'
... Output omitted ...
```

4. Use a logical operator to make a more complex expression that will match the current host in addition to the original host that would have matched.

   4.1. The **or** logical operator will evaluate to **true** if either expression it joins is **true**. Adding the equivalence comparison with **172.25.250.10** will cause the expression to match on **servera**.

```
if $::ipaddress == '172.25.250.10' or $::ipaddress == '172.25.250.11' {
  notify { "Matched." : }
}
```

   4.2. Apply the **regex.pp** manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Matched.
Notice: /Stage[main]/Main/Notify[Matched.]/message: defined 'message'
 as 'Matched.'
Notice: Finished catalog run in 0.05 seconds
```

   4.3. Commit **regex.pp** to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git commit -a -m 'Added logical expression.'
... Output omitted ...
```

5. Modify **regex.pp** to use a regular expression instead of the logical expression to match the IP address.

   5.1. The **=~** operator matches against a regular expression, which is enclosed in slashes (**/ /**).

```
if $::ipaddress =~ /172\.25\.250\.1[01]/ {
  notify { "Matched." : }
}
```

   5.2. Apply the updated **regex.pp** manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Matched.
Notice: /Stage[main]/Main/Notify[Matched.]/message: defined 'message'
 as 'Matched.'
Notice: Finished catalog run in 0.05 seconds
```

The period matches any character in a regular expression. It must be preceded with a backslash to be treated as a literal character. **[01]** matches a single character, which can be either a zero or one.

5.3. Commit **regex.pp** to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git commit -a -m 'Uses a regular expression.'
... Output omitted ...
```

6. Modify **regex.pp** to use a different regular expression that will match the IP address.

6.1. The **\d** expression will match any single digit. The plus is a modifier that will match one or more occurrences of the character that it follows (in this case, digits).

```
if $::ipaddress =~ /172\.25\.250.\d+/ {
  notify { "Matched." : }
}
```

6.2. Apply the updated manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Matched.
Notice: /Stage[main]/Main/Notify[Matched.]/message: defined 'message'
 as 'Matched.'
Notice: Finished catalog run in 0.07 seconds
```

6.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git commit -a -m 'Uses the \d+ regular expression.'
... Output omitted ...
```

7. Insert another regular expression matching string (**[a-z]***) into the original regular expression.

7.1. The resulting manifest should look like the following:

```
if $::ipaddress =~ /172\.25\.250\.[a-z]*\d/ {
  notify { "Matched." : }
}
```

7.2. Apply the updated manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Matched.
Notice: /Stage[main]/Main/Notify[Matched.]/message: defined 'message'
 as 'Matched.'
Notice: Finished catalog run in 0.05 seconds
```

The regular expression still matches the IP address. This is because the asterisk represents zero or more of the characters it follows. In this case, the alphabetic character is optional.

7.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git commit -a -m 'Added [a-z]* expression.'
... Output omitted ...
```

8. Change the asterisk into a plus modifier.

8.1. This will match one or more alphabetic characters, so the IP address will no longer be matched. Modify **regex.pp** to look like the following.

```
if $::ipaddress =~ /172\.25\.250\.[a-z]+\d/ {
  notify { "Matched." : }
}
```

8.2. Apply the updated manifest.

```
[root@servera labwork]# puppet apply regex.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Finished catalog run in 0.05 seconds
```

8.3. Commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git commit -a -m 'Changed [a-z]* to [a-z]+.'
... Output omitted ...
```

9. Examine how the special characters that represent white space and line anchors work in Puppet regular expressions.

9.1. Create a manifest called **redhat.pp** that defines a variable named **$os_name**. The original value of the variable should be the string "Red Hat". The content of the file should look like the following.

```
$os_name = 'Red Hat'

if $os_name =~ /RedHat/ {
  notify { 'RedHat matched.' : }
}

if $os_name =~ /Red\sHat/ {
  notify { 'Red\sHat matched.' : }
}

if $os_name =~ /Red\s*Hat/ {
  notify { 'Red\s*Hat matched.' : }
}

if $os_name =~ /Hat/ {
  notify { 'Hat matched.' : }
}
```

```
if $os_name =~ /^Hat/ {
  notify { '^Hat matched.' : }
}

if $os_name =~ /Hat$/ {
  notify { 'Hat$ matched.' : }
}
```

9.2. Apply the manifest to see which regular expressions match.

```
[root@servera labwork]# puppet apply redhat.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: Hat$ matched.
Notice: /Stage[main]/Main/Notify[Hat$ matched.]/message: defined 'message'
 as 'Hat$ matched.'
Notice: Red\s*Hat matched.
Notice: /Stage[main]/Main/Notify[Red\s*Hat matched.]/message: defined
 'message' as 'Red\s*Hat matched.'
Notice: Hat matched.
Notice: /Stage[main]/Main/Notify[Hat matched.]/message: defined 'message'
 as 'Hat matched.'
Notice: Red\sHat matched.
Notice: /Stage[main]/Main/Notify[Red\sHat matched.]/message: defined
 'message' as 'Red\sHat matched.'
Notice: Finished catalog run in 0.05 seconds
```

9.3. Add the new **redhat.pp** manifest and commit the changes to the local Git repository with a descriptive log message.

```
[root@servera labwork]# git add redhat.pp
[root@servera labwork]# git commit -m 'Initial version.'
... Output omitted ...
```

10. Modify **redhat.pp** and remove the space from the string constant. Apply the manifest again and see which expressions match the modified string.

```
$os_name = 'RedHat'

... Output omitted ...
[root@servera labwork]# puppet apply redhat.pp
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.04 seconds
Notice: RedHat matched.
Notice: /Stage[main]/Main/Notify[RedHat matched.]/message: defined
 'message' as 'RedHat matched.'
Notice: Hat$ matched.
Notice: /Stage[main]/Main/Notify[Hat$ matched.]/message: defined 'message'
 as 'Hat$ matched.'
Notice: Red\s*Hat matched.
Notice: /Stage[main]/Main/Notify[Red\s*Hat matched.]/message: defined
 'message' as 'Red\s*Hat matched.'
Notice: Hat matched.
Notice: /Stage[main]/Main/Notify[Hat matched.]/message: defined 'message'
 as 'Hat matched.'
Notice: Finished catalog run in 0.05 seconds
```

11. Push the contents of the local Git repository to the central repository.

```
[root@servera labwork]# git push
... Output omitted ...
```

Lab: Implementing Variables and Conditionals in a Puppet Module

In this lab, you will write a Puppet manifest that configures **vim** for a user. It will create an SSH **authorized_keys** file for non**root** users.

Resources	
Files	**/etc/puppet/manifests/myenv.pp**
Machines	**workstation** and **servera**

Outcome(s)
You should be able to write a Puppet manifest that uses variables and conditional expressions.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** and run the lab setup script. It checks if the *puppet* package is installed.

```
[student@workstation ~]$ lab puppet-conditionals setup
```

2. Log into **servera** as **root** and create a Puppet manifest called **/etc/puppet/manifests/myenv.pp**.

3. In the manifest, define a variable, **$username**, to specify the user. This user will be the default owner and group owner of files managed by this manifest.

4. Add a conditional statement to set a variable called **$homedir** that stores the home directory of **$username**. The **root** user's home directory is **/root**. Other users' home directories are **/home/*USERNAME***.

5. Add a **file** resource to manage the user's **.vimrc** file and **.vim** directories within their home directory. The **.vim** directory should simply exist. The **.vimrc** file should contain at least the following directives:

```
set ai sw=2
```

6. Add a conditional statement to manage **~/.ssh/authorized_keys** for non**root** users. The **authorized_keys** files should contain the public SSH key for the **student** user on **servera**.

7. Test this manifest as two different users, **root** and **student**, by changing the **$username** variable.

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-conditionals grade
```

9. Once you have successfully finished the lab, copy the working **myenv.pp** manifest into the **labwork** directory and commit it into the local Git repository. Pull (not fetch) upstream changes into the local repository, then push your recent changes to the upstream Git repository.

Solution

In this lab, you will write a Puppet manifest that configures **vim** for a user. It will create an SSH **authorized_keys** file for non**root** users.

Resources	
Files	**/etc/puppet/manifests/myenv.pp**
Machines	**workstation** and **servera**

Outcome(s)
You should be able to write a Puppet manifest that uses variables and conditional expressions.

Before you begin
The *puppet* and *git* packages should already be installed on your **servera** host.

1. Log into **workstation** and run the lab setup script. It checks if the *puppet* package is installed.

```
[student@workstation ~]$ lab puppet-conditionals setup
```

2. Log into **servera** as **root** and create a Puppet manifest called **/etc/puppet/manifests/myenv.pp**.

   If necessary, create the **/etc/puppet/manifests** directory first.

```
[root@servera ~]# mkdir -p /etc/puppet/manifests
[root@servera ~]# vi /etc/puppet/manifests/myenv.pp
```

3. In the manifest, define a variable, **$username**, to specify the user. This user will be the default owner and group owner of files managed by this manifest.

   Add the following lines to **myenv.pp**:

```
$username = 'student'

File {
  owner    => "$username",
  group    => "$username",
}
```

4. Add a conditional statement to set a variable called **$homedir** that stores the home directory of **$username**. The **root** user's home directory is **/root**. Other users' home directories are **/home/*USERNAME***.

   Add the following lines to **myenv.pp**. Be sure to use double quotes around the variable references.

```
if $username == 'root' {

  $homedir = "/$username"

} else {
```

```
  $homedir = "/home/$username"

}
```

5. Add a **file** resource to manage the user's **.vimrc** file and **.vim** directories within their home directory. The **.vim** directory should simply exist. The **.vimrc** file should contain at least the following directives:

```
set ai sw=2
```

Add the following lines to **myenv.pp**. Use the **$username** variable to consistently refer to the user's home directory.

```
file { "/$homedir/.vimrc":
  ensure   => 'file',
  content  => "set ai sw=2\n",
}

file { "/$homedir/.vim":
  ensure   => 'directory',
}
```

6. Add a conditional statement to manage **~/.ssh/authorized_keys** for non**root** users. The **authorized_keys** files should contain the public SSH key for the **student** user on **servera**.

Use **ssh-copy-id** to publish the public key to the **authorized_keys** file.

```
[student@servera ~]$ ssh-copy-id localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 11:92:7e:64:b5:4a:c7:94:1e:ea:c6:62:12:d7:2c:5a.
Are you sure you want to continue connecting (yes/no)? yes
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
filter out any that are already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
student@localhost's password: student

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'localhost'"
and check to make sure that only the key(s) you wanted were added.
```

Display the **authorized_keys** file and create Puppet DSL that will create an exact copy of it to a non**root** user's **authorized_keys** file.

```
[student@servera ~]$ cat ~/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCuzEAZkFuIP85vfeg8Izpu...
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDkXaKlM/8F+b7QIiyS3cRT...
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSBHmVyih/hBbjOcGUGw5y...
```

A great place to put the **file** resource is in the conditional statement that you previously defined. Be sure to make Puppet create the **.ssh** directory first. The resulting Puppet DSL should look something like the following:

```
if $username == 'root' {

  $homedir = "/$username"

  notify { 'No SSH authorized_keys for root': }

} else {

  $homedir = "/home/$username"

  file { "/$homedir/.ssh":
    ensure   => 'directory',
    mode     => '700',
  }

  file { "/$homedir/.ssh/authorized_keys":
    ensure   => 'file',
    content  => ' ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCuzEAZkFuIP85vfeg8Izpu...
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDkXaKlM/8F+b7QIiyS3cRT...
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSBHmVyih/hBbjOcGUGw5y...',
    mode     => '600',
  }

}
```

7. Test this manifest as two different users, **root** and **student**, by changing the **$username** variable.

   First test the manifest for the **root** user.

```
[root@servera ~]# ls -ld ~/.vim ~/.vimrc
[root@servera ~]#
ls: cannot access /root/.vim: No such file or directory
ls: cannot access /root/.vimrc: No such file or directory
[root@servera ~]# grep 'username = ' /etc/puppet/manifests/myenv.pp
$username = 'root',
[root@servera ~]# puppet apply /etc/puppet/manifests/myenv.pp
Notice: Compiled catalog for servera.lab.example.com in environment production in
0.18 seconds
Notice: No SSH authorized_keys for root
Notice: /Stage[main]/Main/Notify[No SSH authorized_keys for root]/message:
defined 'message' as 'No SSH authorized_keys for root'
Notice: /Stage[main]/Main/File[//root/.vimrc]/ensure: defined content as
'{md5}268b95d07e47f072f35cd5055cd5e8dc'
Notice: /Stage[main]/Main/File[//root/.vim]/ensure: created
Notice: Finished catalog run in 0.06 seconds
[root@servera ~]# ls -ld ~/.vim ~/.vimrc
drwxr-xr-x. 2 root root  6 Aug 28 20:40 /root/.vim
-rw-r--r--. 1 root root 12 Aug 28 20:40 /root/.vimrc
[root@servera ~]# cat ~/.vimrc
set ai sw=2
```

   Edit the manifest and change **$username** to **student**. Apply the manifest to make sure it performs the appropriate steps.

```
[root@servera ~]# ls -ld ~student/.vim ~student/.vimrc ~student/.ssh/authorized_keys
ls: cannot access /home/student/.vim: No such file or directory
ls: cannot access /home/student/.vimrc: No such file or directory
-rw-------. 1 student student 413 Aug 28 20:40 /home/student/.ssh/authorized_keys
```

```
[root@servera ~]# grep 'username = ' /etc/puppet/manifests/myenv.pp
$username = 'student'
[root@servera ~]# puppet apply /etc/puppet/manifests/myenv.pp
Notice: Compiled catalog for servera.lab.example.com in environment production in
0.20 seconds
Notice: /Stage[main]/Main/File[//home/student/.vim]/ensure: created
Notice: /Stage[main]/Main/File[//home/student/.vimrc]/ensure: defined
content as '{md5}268b95d07e47f072f35cd5055cd5e8dc'
Notice: /Stage[main]/Main/File[//home/student/.ssh/authorized_keys]/ensure:
defined content as '{md5}16bab87a7a3a8499c752fb20f13201a6'
Notice: Finished catalog run in 0.07 seconds
[root@servera ~]# ls -ld ~student/.vim ~student/.vimrc ~student/.ssh/authorized_keys
-rw-------. 1 student student 413 Aug 28 20:44 /home/student/.ssh/authorized_keys
drwxr-xr-x. 2 student student   6 Aug 28 20:44 /home/student/.vim
-rw-r--r--. 1 student student  12 Aug 28 20:44 /home/student/.vimrc
[root@servera ~]# cat /home/student/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCuzEAZkFuIP85vfeg8Izpu
/Up7LQcuKTU8S4XjL/zuH9ZImWM2Z9otxmBqvw2p+Dp6MJpGbOOYPjWY3mUhyY8gvkTEvd
6wSOt2tv2htrGo3h0OpXyIJVs6l04lLRNVvJ/Eqfk3evt3CeiXMlBUtV6E855VVacwwgX+
PWI3uNN37XBJyi8gkyEV96plTPzohabnUKT4pS0uFK8DbP5WQcg9MSwIyRkA5P8gR4/SkY
/occM+RrymtGWs8JHg3qN9DWysz8epBQmRAKFadGG3okDlxhFhOe6U2qeVr0dpVY3Cwy6M
FQXzZcIjGQw6PkWkebEcp7MSo61ZKGnIdvzNTliB student@servera.lab.example.com
[root@servera ~]# cat /home/student/.vimrc
set ai sw=2
```

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-conditionals grade
```

9. Once you have successfully finished the lab, copy the working **myenv.pp** manifest into the **labwork** directory and commit it into the local Git repository. Pull (not fetch) upstream changes into the local repository, then push your recent changes to the upstream Git repository.

```
[root@servera ~]# cd labwork
[root@servera labwork]# cp /etc/puppet/manifests/myenv.pp .
[root@servera labwork]# git add myenv.pp
[root@servera labwork]# git commit -m 'Working version.'
... Output omitted ...
[root@servera labwork]# git pull
... Output omitted ...
[root@servera labwork]# git push
... Output omitted ...
```

Summary

In this chapter, you learned:

- **`$var = value`** assigns a value to a variable.
- Variables can be referenced using any one of the following forms: **`$var`**, **`$classname::var`**, **`${var}`**, and **`${classname::var}`**.
- Facter facts are global variables that provide system-specific information.
- The **`if-elsif-else`**, **`unless`**, and **`case`** statements can be used to conditionally execute code and apply resources in Puppet manifests.
- Regular expressions in Puppet are enclosed in slashes (**`/ /`**) and are matched with the **`=~`** operator.
- **`\s`** matches white space and **`\d`** matches digits in Puppet regular expressions.
- The asterisk and plus characters are modifiers that multiply the character expression that they immediately follow.
- Selectors (**`? { }`**) are expressions that provide different values based on the value that precedes the question mark.

CHAPTER 10

IDENTIFYING ADVANCED SYSTEM ADMINISTRATION FUNCTIONS IN PUPPET

Overview	
Goal	Identify advanced system administration functions in Puppet code.
Objectives	• Identify advanced system administration functions in Puppet code from OpenStack installers.
Sections	• Identifying Advanced System Administration Functions in Puppet (and Quiz)

Identifying Advanced System Administration Functions in Puppet

Objectives

After completing this section, students should be able to:

- Describe advanced system administration functions in Puppet code from OpenStack installers.

Reviewing Puppet in Red Hat Enterprise Linux OpenStack Platform

As identified earlier, Red Hat Enterprise Linux OpenStack Platform offers a number of installers: **`packstack`**, **`foreman`**, and **`director`**. All use Puppet to install and configure OpenStack services.

Demonstration: Confirm installation of Puppet code from RHEL-OSP

The following steps will demonstrate how to confirm the earlier installation of the Puppet code packages from the RHEL-OSP installers: **`packstack`**, **`foreman`**, and **`director`**.

Resources	
Files	**`/etc/yum.repos.d/rhel7osp.repo`**
Application URL	http://materials.example.com/rhel7osp.repo
Machines	**`workstation`**

Outcome(s)

You should be able to describe how to confirm the RHEL-OSP installers, **`packstack`**, **`foreman`**, and **`director`**, are installed.

1. Verify that the three packages show under the heading of "Installed Packages."

```
[root@workstation ~]# yum list openstack\*puppet\*
... Output omitted ...
Installed Packages
openstack-packstack-puppet.noarch  2015.1-0.11.dev1589.g1d6372f.el7ost @rhel7osp
openstack-puppet-modules.noarch    2015.1.8-8.el7ost                   @rhel7osp
openstack-tripleo-puppet-elements.noarch
                                   0.0.1-4.el7ost                      @rhel7osp
```

2. If, for some reason, the packages are not currently installed, the following pair of commands should rectify that:

```
[root@workstation ~]# wget -P /etc/yum.repos.d/ \
> http://materials.example.com/rhel7osp.repo
... Output omitted ...
[root@workstation ~]# yum -y install openstack\*puppet\*
```

```
... Output omitted ...
```

Reviewing the Puppet code from RHEL-OSP: `config.pp`

The following Puppet code is responsible for creating resources needed to configure the NTP service. It creates one or two file resources: a configuration file and an optional directory.

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/manifests/config.pp
#
class ntp::config inherits ntp {

  if $ntp::keys_enable {
    $directory = ntp_dirname($ntp::keys_file)
    file { $directory:
      ensure => directory,
      owner  => 0,
      group  => 0,
      mode   => '0755',
    }
  }

  file { $ntp::config:
    ensure  => file,
    owner   => 0,
    group   => 0,
    mode    => '0644',
    content => template($ntp::config_template),
  }

}
```

The first resource created by this code is the directory used to store NTP encryption keys. It is only created when the **`$ntp::keys_enable`** variable is set to **`true`**. A new variable, **`$directory`**, is defined to contain the name of the directory to be created. It is derived from the value of the **`$ntp::keys_file`** variable. The **`ntp_dirname()`** function is called and it returns the directory portion of a path name passed as its argument.

The second resource created by this code is the NTP configuration file. It is always created and its name is determined by the value of the **`$ntp::config`** variable. The content of a template file, named by the **`$ntp::config_template`** variable, is transformed into the NTP configuration file.

Where do the variables referenced in the **`ntp::config`** Puppet class get their values? The inherited **`ntp`** class may hold the answer to this question.

Reviewing the Puppet code from RHEL-OSP: `init.pp`

The following Puppet code is the main class that defines the **`ntp`** module. The Puppet class has the same name as the module and is defined in the **`init.pp`** file. It defines the parameters of the Puppet class and calls functions provided by the **`stdlib`** library to validate the parameters passed to the class.

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/manifests/init.pp
class ntp (
  $autoupdate        = $ntp::params::autoupdate,
  $broadcastclient   = $ntp::params::broadcastclient,
  $config            = $ntp::params::config,
  $config_template   = $ntp::params::config_template,
  $disable_auth      = $ntp::params::disable_auth,
```

```
  $disable_monitor   = $ntp::params::disable_monitor,
  $fudge             = $ntp::params::fudge,
  $driftfile         = $ntp::params::driftfile,
  $leapfile          = $ntp::params::leapfile,
  $logfile           = $ntp::params::logfile,
  $iburst_enable     = $ntp::params::iburst_enable,
  $keys_enable       = $ntp::params::keys_enable,
  $keys_file         = $ntp::params::keys_file,
  $keys_controlkey   = $ntp::params::keys_controlkey,
  $keys_requestkey   = $ntp::params::keys_requestkey,
  $keys_trusted      = $ntp::params::keys_trusted,
  $minpoll           = $ntp::params::minpoll,
  $maxpoll           = $ntp::params::maxpoll,
  $package_ensure    = $ntp::params::package_ensure,
  $package_manage    = $ntp::params::package_manage,
  $package_name      = $ntp::params::package_name,
  $panic             = $ntp::params::panic,
  $peers             = $ntp::params::peers,
  $preferred_servers = $ntp::params::preferred_servers,
  $restrict          = $ntp::params::restrict,
  $interfaces        = $ntp::params::interfaces,
  $servers           = $ntp::params::servers,
  $service_enable    = $ntp::params::service_enable,
  $service_ensure    = $ntp::params::service_ensure,
  $service_manage    = $ntp::params::service_manage,
  $service_name      = $ntp::params::service_name,
  $stepout           = $ntp::params::stepout,
  $tinker            = $ntp::params::tinker,
  $udlc              = $ntp::params::udlc,
  $udlc_stratum      = $ntp::params::udlc_stratum,
) inherits ntp::params {

  validate_bool($broadcastclient)
  validate_absolute_path($config)
  validate_string($config_template)
  validate_bool($disable_auth)
  validate_bool($disable_monitor)
  validate_absolute_path($driftfile)
  if $logfile { validate_absolute_path($logfile) }
  if $leapfile { validate_absolute_path($leapfile) }
  validate_bool($iburst_enable)
  validate_bool($keys_enable)
  validate_re($keys_controlkey, ['^\d+$', ''])
  validate_re($keys_requestkey, ['^\d+$', ''])
  validate_array($keys_trusted)
  if $minpoll { validate_numeric($minpoll, 16, 3) }
  if $maxpoll { validate_numeric($maxpoll, 16, 3) }
  validate_string($package_ensure)
  validate_bool($package_manage)
  validate_array($package_name)
  if $panic { validate_numeric($panic, 65535, 0) }
  validate_array($preferred_servers)
  validate_array($restrict)
  validate_array($interfaces)
  validate_array($servers)
  validate_array($fudge)
  validate_bool($service_enable)
  validate_string($service_ensure)
  validate_bool($service_manage)
  validate_string($service_name)
  if $stepout { validate_numeric($stepout, 65535, 0) }
  validate_bool($tinker)
  validate_bool($udlc)
  validate_array($peers)
```

```
  if $autoupdate {
    notice('autoupdate parameter has been deprecated and replaced with package_ensure.
 Set this to latest for the same behavior as autoupdate => true.')
  }

  # Anchor this as per #8040 - this ensures that classes won't float off and
  # mess everything up.  You can read about this at:
  # http://docs.puppetlabs.com/puppet/2.7/reference/lang_containment.html#known-issues
  anchor { 'ntp::begin': } ->
  class { '::ntp::install': } ->
  class { '::ntp::config': } ~>
  class { '::ntp::service': } ->
  anchor { 'ntp::end': }

}
```

The **ntp** class declares and validates the variables used by the **ntp::config** class. Of the four variables that are referenced, all of them are validated except for **$keys_file** (which was referenced as **$ntp::keys_file** in **config.pp**).

This class calls the **validate_absolute_path()**, **validate_string()**, and **validate_bool** **stdlib** library functions to perform the validation.

The main **ntp** class can have the class parameters assigned values when it is referenced. Default values are provided by the **$ntp::params** values that are assigned in the class definition. The inherited **ntp::params** class defines the default values of the referenced variables.

Reviewing the Puppet code from RHEL-OSP: params.pp

The following Puppet code defines the **ntp::params** class. This class defines the default values of the parameters of the **ntp** class. This class uses conditionals based on system facts to determine some default values.

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/manifests/params.pp
class ntp::params {

  $autoupdate        = false
  $config_template   = 'ntp/ntp.conf.erb'
  $disable_monitor   = false
  $keys_enable       = false
  $keys_controlkey   = ''
  $keys_requestkey   = ''
  $keys_trusted      = []
  $logfile           = undef
  $minpoll           = undef
  $leapfile          = undef
  $package_ensure    = 'present'
  $peers             = []
  $preferred_servers = []
  $service_enable    = true
  $service_ensure    = 'running'
  $service_manage    = true
  $stepout           = undef
  $udlc              = false
  $udlc_stratum      = '10'
  $interfaces        = []
  $disable_auth      = false
  $broadcastclient   = false
```

```
  # Allow a list of fudge options
  $fudge             = []

  $default_config       = '/etc/ntp.conf'
  $default_keys_file    = '/etc/ntp/keys'
  $default_driftfile    = '/var/lib/ntp/drift'
  $default_package_name = ['ntp']
  $default_service_name = 'ntpd'

  $package_manage = $::osfamily ? {
    'FreeBSD' => false,
    default   => true,
  }

  if str2bool($::is_virtual) {
    $tinker = true
    $panic  = 0
  }
  else {
    $tinker = false
    $panic  = undef
  }

  case $::osfamily {
    'AIX': {
      $config        = $default_config
      $keys_file     = '/etc/ntp.keys'
      $driftfile     = '/etc/ntp.drift'
      $package_name  = [ 'bos.net.tcp.client' ]
      $restrict      = [
        'default nomodify notrap nopeer noquery',
        '127.0.0.1',
      ]
      $service_name  = 'xntpd'
      $iburst_enable = true
      $servers       = [
        '0.debian.pool.ntp.org',
        '1.debian.pool.ntp.org',
        '2.debian.pool.ntp.org',
        '3.debian.pool.ntp.org',
      ]
      $maxpoll       = undef
    }
... Output omitted ...
    'RedHat': {
      $config          = $default_config
      $keys_file       = $default_keys_file
      $driftfile       = $default_driftfile
      $package_name    = $default_package_name
      $service_name    = $default_service_name
      $restrict        = [
        'default kod nomodify notrap nopeer noquery',
        '-6 default kod nomodify notrap nopeer noquery',
        '127.0.0.1',
        '-6 ::1',
      ]
      $iburst_enable   = false
      $servers         = [
        '0.centos.pool.ntp.org',
        '1.centos.pool.ntp.org',
        '2.centos.pool.ntp.org',
      ]
      $maxpoll         = undef
```

```
    }
... Output omitted ...
    default: {
      fail("The ${module_name} module is not supported on an ${::osfamily} based
 system.")
    }
  }
}
```

The default values for **$config_template** and **$keys_enable** are set early in the Puppet class. They are referenced in the **ntp** class as **$ntp::params::config_template** and **$ntp::params::keys_enable** respectively. The default value defined by **$keys_enable** is **false**, so it must be overridden to use NTP encryption keys.

The value for **$ntp::params::config** is calculated. Initially, a default value for this variable is assigned to the **$default_config** variable. Its final value is assigned in the various clauses of the **case** statement declared in the class. The **case** statement branches based on the value of a **facter** fact, **$::osfamily**.

The value for **$ntp::params::keys_file** is also calculated. When the module is installed on AIX, it gets assigned a default value of **/etc/ntp.keys**. On Red Hat Enterprise Linux, it gets assigned the value of the **$default_keys_file** variable, which is defined as **/etc/ntp/keys**.

Most of the default values are not referenced in resources defined by the Puppet class definitions. Instead, many of them are expanded when the NTP configuration file resource is created from the template file. Consider the following portions of the template file.

```
[root@workstation ~]# cat /usr/share/openstack-puppet/modules/ntp/templates/ntp.conf.erb
# ntp.conf: Managed by puppet.
#
<% if @tinker == true and (@panic or @stepout) -%>
# Enable next tinker options:
# panic - keep ntpd from panicking in the event of a large clock skew
# when a VM guest is suspended and resumed;
# stepout - allow ntpd change offset faster
tinker<% if @panic -%> panic <%= @panic %><% end %><% if @stepout -%> stepout <%=
 @stepout %><% end %>
<% end -%>

... Output omitted ...

# Set up servers for ntpd with next options:
# server - IP address or DNS name of upstream NTP server
# iburst - allow send sync packages faster if upstream unavailable
# prefer - select preferrable server
# minpoll - set minimal update frequency
# maxpoll - set maximal update frequency
<% [@servers].flatten.each do |server| -%>
server <%= server %><% if @iburst_enable == true -%> iburst<% end %><% if
 @preferred_servers.include?(server) -%> prefer<% end %><% if @minpoll -%> minpoll <%=
 @minpoll %><% end %><% if @maxpoll -%> maxpoll <%= @maxpoll %><% end %>
<% end -%>

... Output omitted ...

<% if @keys_enable -%>
keys <%= @keys_file %>
<% unless @keys_trusted.empty? -%>
trustedkey <%= @keys_trusted.join(' ') %>
<% end -%>
```

```
<% if @keys_requestkey != '' -%>
requestkey <%= @keys_requestkey %>
<% end -%>
<% if @keys_controlkey != '' -%>
controlkey <%= @keys_controlkey %>
<% end -%>

<% end -%>
... Output omitted ...
```

If **$ntp::keys_enable** is defined, the **keys** directive is included in the configuration file, pointing to the path name defined in **$ntp::keys_file**. The hosts specified in the **$ntp::servers** list are expanded to separate **server** lines in the resulting NTP configuration file.

References

For more information, see

> Puppet 3.6 Reference Manual
> http://docs.puppetlabs.com/puppet/3.6/reference/

.

Quiz: Identifying Advanced System Administration Functions in Puppet

Use the Puppet class definitions found in the **config.pp**, **init.pp**, and **params.pp** manifests of the **ntp** module definition for Red Hat Enterprise Linux OpenStack Platform. Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. How many **file** resources does the **config.pp** manifest create when default values defined by the module Puppet DSL are used?

   a. Zero.
   b. One.
   c. Two.
   d. Three.

2. Which of the variables are not an inherited variable referenced by the **ntp::config** class definition?

   a. **$config**
   b. **$config_template**
   c. **$directory**
   d. **$keys_file**

3. Which Puppet class defines the default value for the **$servers** variable?

   a. **ntp::config**
   b. **ntp**
   c. **ntp::params**
   d. **init.pp**
   e. **params.pp**

4. Which Puppet classes use class parameters to make decisions? (Choose two.)

   a. **ntp::config**
   b. **ntp**
   c. **ntp::params**
   d. **init.pp**
   e. **params.pp**

5. Which Puppet class uses system facts to make decisions?

   a. **ntp::config**
   b. **ntp**
   c. **ntp::params**

d. **init.pp**
e. **params.pp**

6. Which Puppet class calls **stdlib** library functions to validate class parameters?

a. **ntp::config**
b. **ntp**
c. **ntp::params**
d. **init.pp**
e. **params.pp**

Solution

Use the Puppet class definitions found in the **config.pp**, **init.pp**, and **params.pp** manifests of the **ntp** module definition for Red Hat Enterprise Linux OpenStack Platform. Choose the correct answer to the following questions:

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. How many **file** resources does the **config.pp** manifest create when default values defined by the module Puppet DSL are used?

   a. Zero.
   b. One.
   c. Two.
   d. Three.

2. Which of the variables are not an inherited variable referenced by the **ntp::config** class definition?

   a. `$config`
   b. `$config_template`
   c. `$directory`
   d. `$keys_file`

3. Which Puppet class defines the default value for the **$servers** variable?

   a. `ntp::config`
   b. `ntp`
   c. `ntp::params`
   d. `init.pp`
   e. `params.pp`

4. Which Puppet classes use class parameters to make decisions? (Choose two.)

   a. `ntp::config`
   b. `ntp`
   c. `ntp::params`
   d. `init.pp`
   e. `params.pp`

5. Which Puppet class uses system facts to make decisions?

   a. `ntp::config`
   b. `ntp`
   c. `ntp::params`
   d. `init.pp`
   e. `params.pp`

6. Which Puppet class calls **stdlib** library functions to validate class parameters?

a. `ntp::config`
b. **`ntp`**
c. `ntp::params`
d. `init.pp`
e. `params.pp`

Summary

In this chapter, you learned:

- The **ntp::config** class defines the resources needed to create the configuration files used by the **ntp** service used by RHEL-OSP.
- The **ntp** class, declared in **init.pp**, is the entry point for the **ntp** Puppet module and declares all of the class parameters.
- The **ntp** class also calls **stdlib** library functions to validate the data types of the parameters passed into the module.
- The **ntp::params** class defines the default values for **ntp** class parameters based on system facts.

CHAPTER 11

IMPLEMENTING PUPPET

Overview	
Goal	Deploy and configure a Puppet master and a Puppet client.
Objectives	• Deploy and configure a Puppet master. • Deploy and configure a Puppet client. • Reuse and combine Puppet classes.
Sections	• Implementing a Puppet Master (and Guided Exercise) • Implementing a Puppet Client (and Guided Exercise) • Reusing Puppet Classes (and Guided Exercise)
Lab	• Implementing Puppet

Implementing a Puppet Master

Objectives

After completing this section, students should be able to:

- Deploy and configure a Puppet master.

The Puppet master provides a centralized location for Puppet modules and reports. The Puppet master can build the catalog for each Puppet node based on a configuration stored on the Puppet master. This *node classification* is used to assign classes from modules to Puppet nodes, providing a way to reuse Puppet code on multiple systems. This centralized storage for Puppet modules also provides a means to efficiently update code once and apply it to many Puppet nodes.

Puppet can be run in *standalone mode*; all Puppet nodes have Puppet modules locally that are applied to the system. However, without a centralized Puppet master, it becomes more difficult to manage changes consistently across machines. System administrators have the same issues that they have always had: managing many systems consistently at scale. For this reason and more, it is recommended that a Puppet master be used to store and manage modules for all the Puppet nodes.

In a standalone Puppet configuration, system administrators use the **puppet apply** command directly on the Puppet smoke test file (which includes various classes in the module) to apply changes to the node. In a server/client (master/agent) architecture, the **puppet agent** command can be run from the command-line interface on the Puppet node to immediately apply the classes assigned to the node. Additionally, the **puppet** service should be started and enabled on the Puppet node, which will run **puppet agent** every 30 minutes by default.

Deploying the Puppet master

To deploy a Puppet master, start by installing the *puppet-server* package:

```
[root@host ~]# yum install -y puppet-server
```

Start and enable the **puppetmaster** service:

```
[root@host ~]# systemctl start puppetmaster.service
[root@host ~]# systemctl enable puppetmaster.service
```

The Puppet master listens to TCP/8140, so persistently open that port through the firewall:

```
[root@host ~]# firewall-cmd --permanent --add-port=8140/tcp
```

References

For more information on configuring a Puppet master, see

> Installing Puppet: Post-Install Tasks
> https://docs.puppetlabs.com/guides/install_puppet/post_install.html

For more information on Puppet master architecture, see

> Overview of Puppet's Architecture
> https://docs.puppetlabs.com/puppet/3.6/reference/architecture.html

Guided Exercise: Implementing a Puppet Master

In this lab, you will configure a Red Hat Enterprise Linux server to act as a Puppet master.

Resources	
Machines	**serverc**

Outcome(s)
You should be able to install and launch the Puppet master service on a Red Hat Enterprise Linux 7 server. You should also be able to identify the port the service listens on and open the firewall to accept connections on that port.

Before you begin
The network (host name and IP settings) for the server that will act as the Puppet master should be configured and persistently assigned.

1. Installing a Puppet master begins with installing the Puppet software. Log into **serverc** as **root** and use **yum** to install the *puppet-server* package.

```
[root@serverc ~]# yum -y install puppet-server
```

   The *puppet* package will also be installed as a dependency.

2. Use the **systemctl** command to start the Puppet master service, **puppetmaster.service**. Start the service immediately and persistently.

```
[root@serverc ~]# systemctl start puppetmaster.service
[root@serverc ~]# systemctl enable puppetmaster.service
ln -s '/usr/lib/systemd/system/puppetmaster.service'
 '/etc/systemd/system/multi-user.target.wants/puppetmaster.service'
```

   Use **systemctl status** to confirm that the service started and is running.

```
[root@serverc ~]# systemctl status puppetmaster.service
puppetmaster.service - Puppet master
   Loaded: loaded (/usr/lib/systemd/system/puppetmaster.service; enabled)
   Active: active (running) since Sat 2015-10-03 13:15:06 EDT; 1min 19s ago
 Main PID: 1465 (puppet)
   CGroup: /system.slice/puppetmaster.service
           └─1465 /usr/bin/ruby /usr/bin/puppet master --no-daemonize

Oct 03 13:15:06 serverc.lab.example.com systemd[1]: Starting Puppet master...
Oct 03 13:15:06 serverc.lab.example.com systemd[1]: Started Puppet master.
Oct 03 13:15:10 serverc.lab.example.com puppet-master[1465]: Starting Puppet ...
Hint: Some lines were ellipsized, use -l to show in full.
```

3. Identify the TCP port that the Puppet master is listening to for connections from Puppet clients. The **ss -tlp** command displays information about processes listening on TCP ports.

```
[root@serverc ~]# ss -tlp | grep puppet
LISTEN     0      128                    *:8140                  *:*
 users:(("puppet",1465,9))
```

Puppet is listening on TCP port 8140.

4. Open TCP port 8140 in the firewall. This will allow Puppet clients to connect to the Puppet master.

```
[root@serverc ~]# firewall-cmd --add-port=8140/tcp
success
```

Open the port persistently so it will stay open after a reboot.

```
[root@serverc ~]# firewall-cmd --permanent --add-port=8140/tcp
success
```

Implementing a Puppet Client

Objectives

After completing this section, students should be able to:

- Deploy and configure a Puppet client.

Although Puppet can run in *standalone mode*, where all Puppet clients have Puppet modules locally that are applied to the system, most system administrators find that Puppet works best using a centralized Puppet master. The first step in deploying a Puppet client is to install the *puppet* package:

```
[root@node ~]# yum install -y puppet
```

Once the *puppet* package is installed, the Puppet client must be configured with the host name of the Puppet master. The host name of the Puppet master should be placed in the **/etc/puppet/puppet.conf** file, under the **[agent]** section. The following snippet shows an example where the Puppet master is named **puppet.lab.example.com**:

```
[agent]
    server = puppet.lab.example.com
```

Note

When an explicit Puppet master is not defined, Puppet uses a default host name of **puppet**. If the DNS search path includes a host named **puppet**, this host will be used automatically.

The final step to be taken on the Puppet client is to start the Puppet agent service and configure it to run at boot time.

```
[root@node ~]# systemctl start puppet.service
[root@node ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service'
 '/etc/systemd/system/multi-user.target.wants/puppet.service'
```

The Puppet agent service will generate a host certificate and send a certificate-signing request to the Puppet master. Once the client certificate request has been sent, the request to sign the client certificate can be seen on the Puppet master:

```
[root@master ~]# puppet cert list
  "node.example.com" (SHA256) 24:C6:06:B5:43:D8:4C:77:C8:F6:68:9B:4C:8D:A9:29:74:
 05:70:01:0D:DB:D8:C0:10:D2:4D:FF:F3:A2:3B:32
```

The **puppet cert sign** command will sign the client certificate on the Puppet master:

```
[root@master ~]# puppet cert sign node.example.com.
Notice: Signed certificate request for node.example.com
Notice: Removing file Puppet::SSL::CertificateRequest node.example.com
```

```
at '/var/lib/puppet/ssl/ca/requests/node.example.com.pem'
```

The **puppet cert list** command only shows pending requests. Thus, it should report no pending keys to be signed.

```
[root@master ~]# puppet cert list
```

To view an already-signed certificate, append the host name after the command.

```
[root@master ~]# puppet cert list node.example.com
+ "node.example.com" (SHA256) 64:C7:00:BB:DB:4C:74:1F:CD:D5:B2:83:87:53:65:28:32:
 32:25:A3:2B:A7:51:06:71:7E:A5:69:BB:5E:D3:33
```

Once the certificate has been signed by the Puppet master, the Puppet agent will receive a catalog and apply any changes needed.

References

For more information on configuring a Puppet client, see

> Installing Puppet: Post-Install Tasks
> https://docs.puppetlabs.com/guides/install_puppet/post_install.html

Guided Exercise: Implementing a Puppet Client

In this lab, you will configure a Red Hat Enterprise Linux server to act as a Puppet client.

Resources	
Files	**/etc/puppet/puppet.conf**
Machines	**serverb** and **serverc**

Outcome(s)
You should be able to install and activate a Puppet client on a Red Hat Enterprise Linux 7 server.

Before you begin
The network (host name and IP settings) for the server that will act as the Puppet client should be configured and persistently assigned.

1. Installing a Puppet client begins with installing the Puppet software. Log into **serverb** as **root** and install the *puppet* package.

```
[root@serverb ~]# yum -y install puppet
```

2. Configure the Puppet agent to reference the Puppet master. Edit **/etc/puppet/puppet.conf** and add the following content to the **[agent]** stanza of the configuration file.

```
    # Define which Puppet master to use.
    server = serverc.lab.example.com
```

The **server** setting should point to the fully qualified host name of the Puppet master; in our case, **serverc.lab.example.com**.

3. Normally you would start the Puppet agent service and configure it to run at boot time. In this practice exercise, you will run the Puppet agent by hand so you can see its interaction with the Puppet master.

The first time **puppet agent** runs on a Puppet client, it generates a certificate and sends this certificate to the Puppet master. The Puppet master must accept, sign, and return this certificate back to the client before it is allowed to get a catalog from the Puppet master. When **puppet agent** first runs, it waits for a default two minutes (120 seconds) and will resend the request to sign the certificate. The Puppet master can be configured to autosign certificates, but that poses a security risk, so normally a client certificate must be manually signed on the Puppet master.

Run the **puppet agent** command so that it resends a certificate-signing request every 10 seconds until it receives a response from the Puppet master. The **--waitforcert** option changes the time between resending the certificate request.

```
[root@serverb ~]# puppet agent --waitforcert 10
```

A look at **/var/log/messages** will show the agent's retries.

```
[root@serverb ~]# tail /var/log/messages
... Output omitted ...
Oct 26 18:48:25 localhost puppet-agent[1493]: Did not receive certificate
Oct 26 18:48:35 localhost puppet-agent[1493]: Did not receive certificate
Oct 26 18:48:45 localhost puppet-agent[1493]: Did not receive certificate
```

4. Log into the Puppet master, **serverc**, as **root**. Use the **puppet cert list** command to list the pending certificate-signing requests from Puppet nodes.

```
[root@serverc ~]# puppet cert list
  "serverb.lab.example.com" (SHA256) 25:03:59:01:D9:33:7B:5D:8D:E0:C2:53:41:5D:CE:
 00:A1:0D:2D: CA:AA:08:7C:5F:AE:17:3C:37:14:6C:1A:62
```

5. Sign the client certificate on the Puppet master.

```
[root@serverc ~]# puppet cert sign serverb.lab.example.com
Notice: Signed certificate request for serverb.lab.example.com
Notice: Removing file Puppet::SSL::CertificateRequest serverb.lab.example.com at
'/var/lib/puppet/ssl/ca/requests/serverb.lab.example.com.pem'
```

6. Listing the pending certificate requests on the Puppet master will not display any output. Adding the host name of the Puppet client to the **puppet cert list** command will list the fingerprint of its signed certificate.

```
[root@serverc ~]# puppet cert list
[root@serverc ~]# puppet cert list serverb.lab.example.com
+ "serverb.lab.example.com" (SHA256) 4B:9F:60:B0:A6:89:26:09:B0:72:6A:9A:FF:46:E5:
 EE:F7:E2:C8:F2:12:62:67:44:69:C7:10:D5:B0:D9:67:18
```

7. On the Puppet client, **serverb**, check **/var/log/messages** and confirm the Puppet agent received the signed certificate. The output should look like the following:

```
[root@serverb ~]# tail /var/log/messages
... Output omitted ...
Oct 26 18:50:15 localhost puppet-agent[1493]: Did not receive certificate
Oct 26 18:50:25 localhost puppet-agent[1493]: Did not receive certificate
Oct 26 18:50:35 localhost puppet-agent[1493]: Starting Puppet client version 3.6.2
Oct 26 18:50:36 localhost puppet-agent[26188]: Finished catalog run in 0.05 seconds
```

When the certificate is received, the original **puppet agent** process will spawn a child process to accept and run a catalog from the Puppet master.

8. On the Puppet client, run the Puppet agent with the **--noop** option to make sure it works with the Puppet master. If all goes well, you should see output similar to the following.

```
[root@serverb ~]# puppet agent --test --noop
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
Info: Applying configuration version '1445899836'
Notice: Finished catalog run in 0.02 seconds
```

9. The Puppet agent that was launched initially is still running in the background. It needs to be identified and killed before the Puppet agent service can be launched.

```
[root@serverb ~]# ps -ef | grep puppet
root      1493     1  0 13:34 ?        00:00:04 /usr/bin/ruby /usr/bin/puppet agent
 --waitforcert 10
root     26431 25952  0 15:52 pts/0    00:00:00 grep --color=auto puppet
[root@serverb ~]# kill 1493
[root@serverb ~]# ps -ef | grep puppet
root     26435 25952  0 15:53 pts/0    00:00:00 grep --color=auto puppet
```

10. Use the **systemctl** command to start and enable the **puppet.service** service. Check its status to confirm the service is running.

```
[root@serverb ~]# systemctl start puppet.service
[root@serverb ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service' '/etc/systemd/system/multi-
user.target.wants/puppet.service'
[root@serverb ~]# systemctl status puppet.service
puppet.service - Puppet agent
   Loaded: loaded (/usr/lib/systemd/system/puppet.service; enabled)
   Active: active (running) since Mon 2015-10-26 15:53:38 PDT; 12s ago
 Main PID: 26438 (puppet)
   CGroup: /system.slice/puppet.service
           └─26438 /usr/bin/ruby /usr/bin/puppet agent --no-daemonize

Oct 26 15:53:38 serverb.lab.example.com systemd[1]: Starting Puppet agent...
Oct 26 15:53:38 serverb.lab.example.com systemd[1]: Started Puppet agent.
Oct 26 15:53:39 serverb.lab.example.com puppet-agent[26438]: Starting Puppet ...
Oct 26 15:53:39 serverb.lab.example.com puppet-agent[26444]: Finished catalog...
Hint: Some lines were ellipsized, use -l to show in full.
```

Reusing Puppet Classes

Objectives

After completing this section, students should be able to:

- Reuse and combine Puppet classes.

Once the Puppet master is deployed, install modules on the Puppet master using the **`puppet module install`** command. The classes in the modules can now be assigned to Puppet nodes using *node classification*. Node classification is configured in the **`/etc/puppet/manifests/site.pp`** file. This file should always include a **`default`** section, even if there are no classes defined therein, to avoid misclassification. The following example shows a **`default`** section including the **`test`** class (presumably defined in an installed module):

```
node default {
  include test
}
```

Notice that the syntax is fairly familiar, as it uses a name and curly braces ({}) to contain the definition. A node name (matching the host name on the certificate) can be specified to include a class. The following example shows an empty default node definition, followed by a definition for **`node.example.com`** using the **`test`** class:

```
node default {
}
node 'node.example.com' {
 include test
}
```

Note

The **`default`** name is a special keyword, and should not be quoted. The host names are strings, and should always be quoted.

Multiple nodes can be specified in a comma-separated list:

```
node default {
}
node 'node1.example.com', 'node2.example.com', 'node3.example.com' {
 include test
}
```

Note

Try to avoid multiple matches for a node name because it may provide unexpected results. Node classification is matched using the first most specific match, so file order may make a difference.

Regular expressions can also be used to match nodes. The word matches (**^** for beginning of word and **$** for end of word) are often used to limit the extent of the match. The following example matches the three nodes in the previous example:

```
node default {
}
node /^node[1-3].example.com$/ {
 include test
}
```

The **\d** selector can be used to match any number. The **+** modifier is used to match one or more of the previous character. Thus, when they are used together (as in the following example), the grouping will match one or more numbers:

```
node default {
}
node /^node\d+.example.com$/ {
 include test
}
```

This regular expression would match the following host names (and many more):

- node1.example.com
- node2.example.com
- node10.example.com
- node42.example.com
- node123456.example.com

Once the Puppet node has a node classification, the next run will perform the steps necessary to bring the node into alignment with the catalog. When the **puppet** service first runs, it will check in with the Puppet master (requesting a certificate if necessary), and check in every 30 minutes after that by default. This default of 30 minutes can be changed using the **runinterval** option in the **/etc/puppet/puppet.conf** file. The value is normally given in seconds, but a modifier can be appended such as **m** for minutes or **h** for hours. The following shows a snippet of **/etc/puppet/puppet.conf** that uses a value of 15 minutes instead of the default 30.

```
[agent]
  runinterval = 15m
```

Once this option is set, Puppet agent will automatically detect the change and begin connecting to the Puppet master every 15 minutes; there is no need to restart the service to pick up the change. To verify the configuration, use the **--configprint** option to the **puppet agent** command:

```
[root@host ~]# puppet agent --configprint runinterval
900
```

Note

The **`puppet agent --configprint`** command must be run as the **`root`** user to get an accurate value. If the command is run as a normal, nonprivileged user, the built-in defaults will be reported, not the currently active values.

References

For more information on Puppet node classification, see

> Language: Node Definitions
> https://docs.puppetlabs.com/puppet/3.6/reference/lang_node_definitions.html

Guided Exercise: Reusing Puppet Classes

In this lab, you will create a new module that uses multiple Puppet classes to deploy a web server with multiple administrators. You will use a module, written in a previous lab, as the basis for the new module. It will include a new Puppet class, called **webadmins**, that will ensure multiple web administrator accounts are on the system.

Resources	
Files	The **rht-ftpcustom** module.
Machines	**servera**, **serverb**, **serverc**

Outcome(s)
You should be able to assign classification to a node.

Before you begin
The **rht-ftpcustom** module should already be available on your **servera** host. You should have **serverc** configured as a Puppet master, and **serverb** configured as a Puppet node.

1. As **student** on **servera**, copy the **rht-ftpcustom** module from **servera** to **serverc**.

```
[student@servera ~]$ scp labwork/rht-ftpcustom/pkg/rht-ftpcustom-0.1.0.tar.gz
 root@serverc:
```

2. As **root** on **serverc**, install the **rht-ftpcustom** module.

```
[root@serverc ~]# puppet module install rht-ftpcustom-0.1.0.tar.gz
```

3. On **serverb**, verify the **ftpcustom** class will be included in a **puppet agent** run, but do not apply the changes.

```
[root@serverb ~]# puppet agent --test --noop
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
... Output omitted ...
Notice: Class[Ftpcustom::Install]: Would have triggered 'refresh' from 1 events
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/ensure:
 current_value absent, should be file (noop)
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/custom-banner]/ensure:
 current_value absent, should be file (noop)
Notice: Class[Ftpcustom::Config]: Would have triggered 'refresh' from 2 events
Info: Class[Ftpcustom::Config]: Scheduling refresh of Class[Ftpcustom::Service]
Notice: Class[Ftpcustom::Service]: Would have triggered 'refresh' from 1 events
Info: Class[Ftpcustom::Service]: Scheduling refresh of Service[vsftpd]
Notice: /Stage[main]/Ftpcustom::Service/Service[vsftpd]/ensure: current_value
 stopped, should be running (noop)
Info: /Stage[main]/Ftpcustom::Service/Service[vsftpd]: Unscheduling refresh on
 Service[vsftpd]
Notice: Class[Ftpcustom::Service]: Would have triggered 'refresh' from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 3 events
Notice: Finished catalog run in 0.71 seconds
```

4. On **serverc**, edit the **/etc/puppet/manifests/site.pp** file again. Comment out the **ftpcustom** class in the default section and include it in a new node classification that includes **servera** and **serverb**. Note that host name strings should be quoted. The file should contain the following:

```
node default {
  #include ftpcustom
}

node 'servera.lab.example.com', 'serverb.lab.example.com' {
  include ftpcustom
}
```

5. On **serverb**, verify the **ftpcustom** class will be included in a **puppet agent** run, but do not apply the changes.

```
[root@serverb ~]# puppet agent --test --noop
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
... Output omitted ...
Notice: Class[Ftpcustom::Install]: Would have triggered 'refresh' from 1 events
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/ensure:
 current_value absent, should be file (noop)
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/custom-banner]/ensure:
 current_value absent, should be file (noop)
Notice: Class[Ftpcustom::Config]: Would have triggered 'refresh' from 2 events
Info: Class[Ftpcustom::Config]: Scheduling refresh of Class[Ftpcustom::Service]
Notice: Class[Ftpcustom::Service]: Would have triggered 'refresh' from 1 events
Info: Class[Ftpcustom::Service]: Scheduling refresh of Service[vsftpd]
Notice: /Stage[main]/Ftpcustom::Service/Service[vsftpd]/ensure: current_value
 stopped, should be running (noop)
Info: /Stage[main]/Ftpcustom::Service/Service[vsftpd]: Unscheduling refresh on
 Service[vsftpd]
Notice: Class[Ftpcustom::Service]: Would have triggered 'refresh' from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 3 events
Notice: Finished catalog run in 0.71 seconds
```

6. On **serverc**, edit the **/etc/puppet/manifests/site.pp** file one last time. Include **serverc** in the node classification this time. Remove the individual node list and use a regular expression to match all three servers. The file should contain the following:

```
node default {
  #include ftpcustom
}

node /^server[a-c].lab.example.com$/ {
  include ftpcustom
}
```

7. On **serverb**, verify the **ftpcustom** class is included in a **puppet agent** run. Apply the changes this time.

```
[root@serverb ~]# puppet agent --test
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
...output omitted...
Notice: /Stage[main]/Ftpcustom::Install/Package[vsftpd]/ensure: created
```

```
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/content:
--- /etc/vsftpd/vsftpd.conf 2014-03-07 04:58:21.000000000 -0500
+++ /tmp/puppet-file20151016-27916-17s23ed 2015-10-16 15:44:42.498593814 -0400
@@ -84,6 +84,7 @@
 #
 # You may fully customise the login banner string:
 #ftpd_banner=Welcome to blah FTP service.
+banner_file=/etc/vsftpd/custom-banner
 #
 # You may specify a file of disallowed anonymous e-mail addresses. Apparently
 # useful for combatting certain DoS attacks.

Info: FileBucket got a duplicate file {md5}c4072ca90053a6e86cf86850c343346d
Info: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]: Filebucketed /
etc/vsftpd/vsftpd.conf to puppet with sum c4072ca90053a6e86cf86850c343346d
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/vsftpd.conf]/
content: content changed '{md5}c4072ca90053a6e86cf86850c343346d' to
 '{md5}81ee930dab4087a96ea4f55b699702d2'
Notice: /Stage[main]/Ftpcustom::Config/File[/etc/vsftpd/custom-banner]/ensure:
 defined content as '{md5}61c72b9a1e7bf54059d3b680dd1a0929'
Info: Class[Ftpcustom::Config]: Scheduling refresh of Class[Ftpcustom::Service]
Info: Class[Ftpcustom::Service]: Scheduling refresh of Service[vsftpd]
Notice: /Stage[main]/Ftpcustom::Service/Service[vsftpd]/ensure: ensure changed
 'stopped' to 'running'
Info: /Stage[main]/Ftpcustom::Service/Service[vsftpd]: Unscheduling refresh on
 Service[vsftpd]
Notice: Finished catalog run in 3.92 seconds
```

8. Verify the functionality of the FTP server.

   8.1. Install the FTP client.

   ```
   [root@serverb ~]# yum install -y ftp
   ```

   8.2. Connect to the local FTP server.

   ```
   [root@serverb ~]# ftp ::1
   Connected to ::1 (::1).
   220-###############################################################
   220-#################### Private FTP Server ######################
   220-############# Only authorized users permitted. #############
   220-###############################################################
   220
   Name (::1:root):
   ```

   8.3. Once you have confirmed the custom banner is displayed, use **Ctrl**+**C** to break out and get back to the shell.

Lab: Implementing Puppet

In this lab, you will configure **serverc** to act as a Puppet master. You will also configure **serverb** to act as a Puppet client.

Resources	
Files	**/etc/puppet/puppet.conf**, **/etc/puppet/manifests/site.pp**
Application URL	**http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**
Machines	**serverb** and **serverc**

Outcome(s)
You should be able to install and launch the Puppet master service and install and activate a Puppet client on Red Hat Enterprise Linux 7 servers. You should also be able to install a module on the Puppet master and apply the changes to the Puppet node(s).

Before you begin
Reset both your **serverb** and **serverc** servers. The network (host name and IP settings) for the servers should be configured and persistently assigned.

You have been given two newly installed machines: **serverb.lab.example.com** and **serverc.lab.example.com**. Deploy **serverc** as the Puppet master, and **serverb** as the Puppet node. Install the **thoraxe-motd** Puppet module found at **http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz** on the Puppet master. Configure the node classification on the Puppet master so it uses the **motd** class from this module on **servera.lab.example.com**, **serverb.lab.example.com**, and **serverc.lab.example.com**. Apply the class to the **serverb.lab.example.com** system and verify the **/etc/motd** file.

1. Log into **workstation** as **student** and run the lab setup script. It confirms that the host name and network settings of the Puppet master and client server(s) are correct.

```
[student@workstation ~]$ lab puppet-master-client setup
```

2. Install the Puppet master software on **serverc**.

3. Start the appropriate service and configure it to run at boot time. Make sure any firewall ports have been opened so clients can connect to the Puppet service.

4. Install the Puppet agent software on **serverb**.

5. Configure the Puppet agent to reference the Puppet master, **serverc.lab.example.com**.

6. Start the appropriate Puppet client daemon and configure it to run at boot time. It will generate a client host certificate and send a certificate-signing request to the Puppet master.

7. Sign the client's certificate on the Puppet master.

8. Confirm the Puppet client is working properly with the Puppet master.

9. On **serverc**, download the **motd** Puppet module from **http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**.

10. Install the **thoraxe-motd** module.

11. On **serverc**, create the **/etc/puppet/manifests/site.pp** file and add a node classification that includes the **motd** class from the **thoraxe-motd** module for **servera**, **serverb**, and **serverc**.

12. On **serverb**, verify the **motd** class will be included in a **puppet agent** run, but do not apply the changes.

13. Apply the Puppet class to **serverb**.

14. Verify the addition to **serverb**.

15. Run the lab grading script from **workstation** to check your work. The grading program will access the Puppet master and client remotely to evaluate your work.

```
[student@workstation ~]$ lab puppet-master-client grade
```

Solution

In this lab, you will configure **serverc** to act as a Puppet master. You will also configure **serverb** to act as a Puppet client.

Resources	
Files	**/etc/puppet/puppet.conf**, **/etc/puppet/manifests/site.pp**
Application URL	**http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**
Machines	**serverb** and **serverc**

Outcome(s)
You should be able to install and launch the Puppet master service and install and activate a Puppet client on Red Hat Enterprise Linux 7 servers. You should also be able to install a module on the Puppet master and apply the changes to the Puppet node(s).

Before you begin
Reset both your **serverb** and **serverc** servers. The network (host name and IP settings) for the servers should be configured and persistently assigned.

You have been given two newly installed machines: **serverb.lab.example.com** and **serverc.lab.example.com**. Deploy **serverc** as the Puppet master, and **serverb** as the Puppet node. Install the **thoraxe-motd** Puppet module found at **http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz** on the Puppet master. Configure the node classification on the Puppet master so it uses the **motd** class from this module on **servera.lab.example.com**, **serverb.lab.example.com**, and **serverc.lab.example.com**. Apply the class to the **serverb.lab.example.com** system and verify the **/etc/motd** file.

1. Log into **workstation** as **student** and run the lab setup script. It confirms that the host name and network settings of the Puppet master and client server(s) are correct.

```
[student@workstation ~]$ lab puppet-master-client setup
```

2. Install the Puppet master software on **serverc**.

   Log in as **root** on **serverc** and use **yum** to install the *puppet-server* package.

```
[root@serverc ~]# yum -y install puppet-server
```

3. Start the appropriate service and configure it to run at boot time. Make sure any firewall ports have been opened so clients can connect to the Puppet service.

   3.1. Use the **systemctl** command to start the Puppet master service, **puppetmaster.service**. Start the service immediately and persistently.

```
[root@serverc ~]# systemctl start puppetmaster.service
[root@serverc ~]# systemctl enable puppetmaster.service
ln -s '/usr/lib/systemd/system/puppetmaster.service'
 '/etc/systemd/system/multi-user.target.wants/puppetmaster.service'
```

3.2. The Puppet server service listens on TCP port 8140. Open this port in the firewall persistently so Puppet clients can connect to the Puppet master.

```
[root@serverc ~]# firewall-cmd --add-port=8140/tcp
success
[root@serverc ~]# firewall-cmd --permanent --add-port=8140/tcp
success
```

4. Install the Puppet agent software on **serverb**.

Log into **serverb** as **root** and use **yum** to install the *puppet* package.

```
[root@serverb ~]# yum -y install puppet
```

5. Configure the Puppet agent to reference the Puppet master, **serverc.lab.example.com**.

Edit the **/etc/puppet/puppet.conf** configuration file and add the following content to the **[agent]** stanza.

```
    # Define which Puppet master to use.
    server = serverc.lab.example.com
```

6. Start the appropriate Puppet client daemon and configure it to run at boot time. It will generate a client host certificate and send a certificate-signing request to the Puppet master.

```
[root@serverb ~]# systemctl start puppet.service
[root@serverb ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service'
 '/etc/systemd/system/multi-user.target.wants/puppet.service'
```

7. Sign the client's certificate on the Puppet master.

7.1. Go back to the Puppet master and use the **puppet cert list** command to list pending certificate-signing requests from Puppet clients.

```
[root@serverc ~]# puppet cert list
  "serverb.lab.example.com" (SHA256) 24:C6:06:B5:43:D8:4C:77:C8:F6:68:9B:4C:8D:
  A9:29:74:05:70:01:0D:DB:D8:C0:10:D2:4D:FF:F3:A2:3B:32
```

7.2. Sign the certificate-signing request from the client, **serverb**.

```
[root@serverc ~]# puppet cert sign serverb.lab.example.com
Notice: Signed certificate request for serverb.lab.example.com
Notice: Removing file Puppet::SSL::CertificateRequest serverb.lab.example.com
 at '/var/lib/puppet/ssl/ca/requests/serverb.lab.example.com.pem'
```

8. Confirm the Puppet client is working properly with the Puppet master.

Run the Puppet agent in **noop** mode on the Puppet client to make sure it works with the Puppet master.

```
[root@serverb ~]# puppet agent --test --noop
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
Info: Applying configuration version '1443898587'
Notice: Finished catalog run in 0.06 seconds
```

9. On **serverc**, download the **motd** Puppet module from **http://materials.example.com/modules/thoraxe-motd-0.1.1.tar.gz**.

```
[root@serverc ~]$ wget http://materials.example.com/modules/thoraxe-
motd-0.1.1.tar.gz
```

10. Install the **thoraxe-motd** module.

```
[root@serverc ~]# puppet module install thoraxe-motd-0.1.1.tar.gz
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── thoraxe-motd (v0.1.1)
```

11. On **serverc**, create the **/etc/puppet/manifests/site.pp** file and add a node classification that includes the **motd** class from the **thoraxe-motd** module for **servera**, **serverb**, and **serverc**.

    There are several ways to accomplish this task; this example uses the default node definition, but you could have used a comma-separated list for the servers, or a regular expression that matched the server names. The file should contain the following:

```
node default {
  include motd
}
```

12. On **serverb**, verify the **motd** class will be included in a **puppet agent** run, but do not apply the changes.

```
[root@serverb ~]# puppet agent --test --noop
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
Info: Applying configuration version '1445026071'
Notice: /Stage[main]/Motd/File[/etc/motd]/content:
--- /etc/motd 2013-06-07 10:31:32.000000000 -0400
+++ /tmp/puppet-file20151016-28316-189cjl1 2015-10-16 16:07:52.092218383 -0400
@@ -0,0 +1 @@
+This is the default message

Notice: /Stage[main]/Motd/File[/etc/motd]/content:
 current_value {md5}d41d8cd98f00b204e9800998ecf8427e, should be
 {md5}c01d1bd4d04a962e8364a4fb55e59f05 (noop)
Notice: Class[Motd]: Would have triggered 'refresh' from 1 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.11 seconds
```

13. Apply the Puppet class to **serverb**.

```
[root@serverb ~]$ puppet agent --test
Info: Retrieving plugin
Info: Caching catalog for serverb.lab.example.com
Info: Applying configuration version '1445026071'
Notice: /Stage[main]/Motd/File[/etc/motd]/content:
--- /etc/motd 2013-06-07 10:31:32.000000000 -0400
+++ /tmp/puppet-file20151016-28447-8tzqi3 2015-10-16 16:09:49.094928524 -0400
@@ -0,0 +1 @@
+This is the default message

Info: /Stage[main]/Motd/File[/etc/motd]: Filebucketed /etc/motd to puppet with sum
 d41d8cd98f00b204e9800998ecf8427e
Notice: /Stage[main]/Motd/File[/etc/motd]/content: content changed
 '{md5}d41d8cd98f00b204e9800998ecf8427e' to '{md5}c01d1bd4d04a962e8364a4fb55e59f05'
Notice: Finished catalog run in 0.17 seconds
```

14. Verify the addition to **serverb**.

```
[root@serverb ~]$ cat /etc/motd
This is the default message
```

15. Run the lab grading script from **workstation** to check your work. The grading program will access the Puppet master and client remotely to evaluate your work.

```
[student@workstation ~]$ lab puppet-master-client grade
```

Summary

In this chapter, you learned:

- The *puppet-server* package must be installed to implement a Puppet master.
- The **puppetmaster.service** service manages the Puppet master daemon.
- TCP port 8140 must be opened on a server that acts as a Puppet master.
- The *puppet* package includes all the software needed to implement a Puppet client.
- The **server = FQDN-of-master** line of the **[agent]** section of **/etc/puppet/puppet.conf** points the Puppet agent software to the Puppet master.
- The **puppet.service systemd** service controls the Puppet agent software on a client.
- When a Puppet agent checks in with a Puppet master, the **puppet cert sign FQDN-of-client** command signs the client's CSR and returns the signed host certificate at the next check-in.
- Node definitions in **/etc/puppet/manifests/site.pp** define the Puppet classes that will be deployed on Puppet clients.

CHAPTER 12

IMPLEMENTING EXTERNAL PUPPET MODULES

Overview	
Goal	Implement Puppet modules from Puppet Forge.
Objectives	• Locate Puppet modules in Puppet Forge. • Implement a Puppet module from Puppet Forge.
Sections	• Puppet Files in Puppet Forge (and Quiz) • Implementing a Module from Puppet Forge (and Guided Exercise)
Lab	• Implementing External Puppet Modules

Puppet Modules in Puppet Forge

Objectives

After completing this section, students should be able to:

- Locate Puppet modules in Puppet Forge.

Puppet Forge [https://forge.puppetlabs.com] is a public library of Puppet modules written by a variety of Puppet administrators and users. It is an archive that contains thousands of Puppet modules and it has a searchable database that helps Puppet users identify modules that might help them accomplish an administrative task. Puppet Forge includes links to documentation for new Puppet module developers. It also includes links to various tools that will facilitate the writing of Puppet modules.

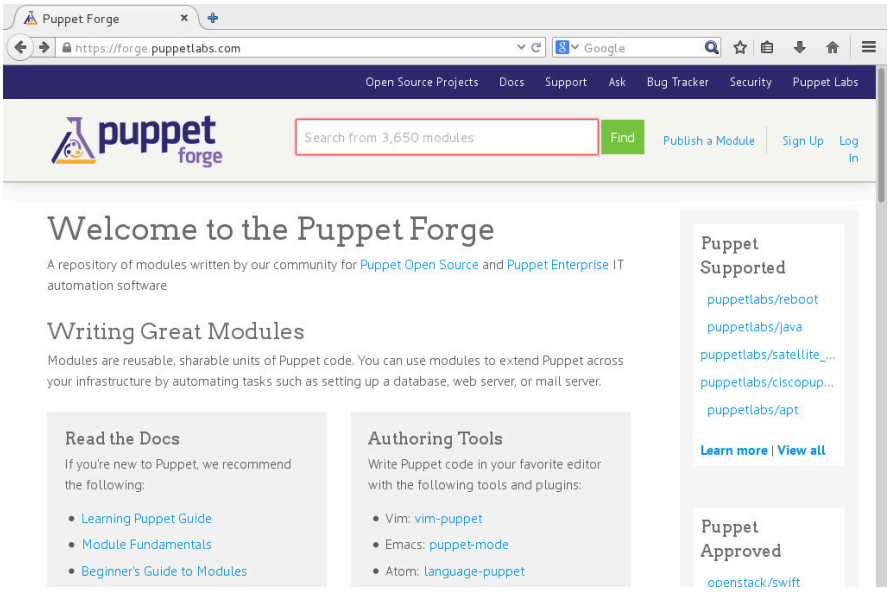

Figure 12.1: Puppet Forge home page

Approved modules

Some of the modules that are published on Puppet Forge are marked as "approved" modules. These are modules that Puppet Labs has determined are useful, well written, well documented, and provide metadata that includes license and contact information.

Approved modules are actively being developed and enhanced. They have been checked and found to be free from harmful code. Module authors and contributors provide informal support for approved modules. No formal support is provided for these modules.

Supported modules

"Supported" modules are modules that Puppet Labs provides commercial support for. They have been tested to work with their Puppet Enterprise product on a variety of platforms. Bug reports are encouraged and there is a link to a JIRA page that helps Puppet Labs manage the maintenance of these modules.

Benefits of Puppet Forge

Puppet Forge helps developers apply open source principles to Puppet module development. If an administrator has a system configuration need that Puppet can solve, there is often no need to start a new module from scratch. They can search Puppet Forge to see if someone else has published a module that does the job.

Puppet Forge allows several administrators and developers to collaborate. They can work as a team to develop, test, and document a module that they all find useful. That kind of collaboration benefits the whole DevOps community.

Anyone can create a Puppet Forge account at the *Puppet Forge Sign Up* [https://forge.puppetlabs.com/signup] page. An account is not necessary to use Puppet Forge, but it is required if a developer wants to publish their own Puppet modules for use by others.

The following information must be provided when creating a Puppet Forge account:

- **Username**—An alphanumeric name that will be used as the author portion of module names.
- **Display Name**—Usually the full name of the user who owns the account. It is the name that the other users of Puppet Forge will see.
- **Email**—The email address used to contact the account owner.
- **Password** (and **Confirmation**)—The authentication password used by the owner to access the account.

Searching Puppet Forge for modules

The Puppet Forge home page has a search box at the top of the page. Specified search terms are compared against module names, descriptions, and tags that may be assigned to them. When matches are found, a list of module authors and names are presented. The names are hyperlinks that go to pages with more detailed information about the module or author, depending on the link that is followed.

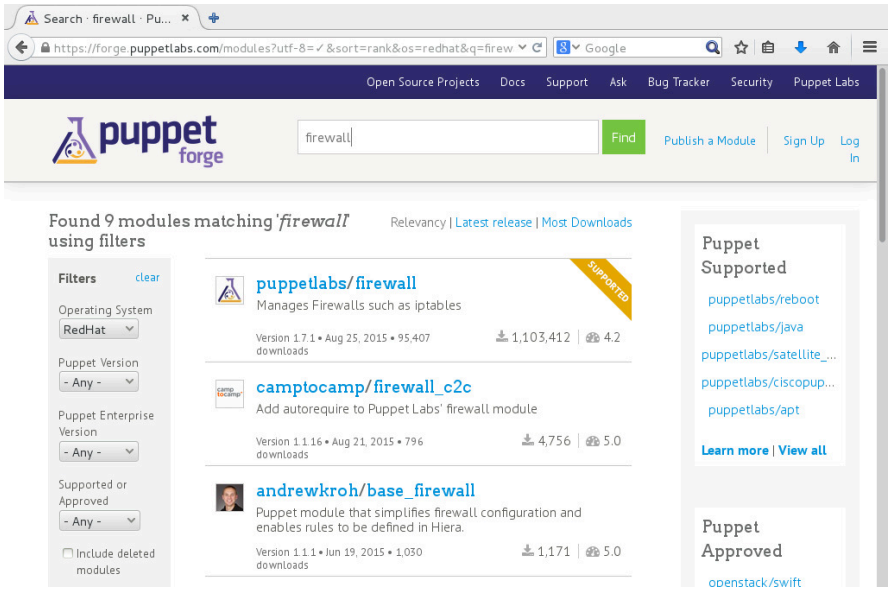

Figure 12.2: Results of Puppet Forge search

Each module is assigned a quality score by Puppet Labs. Other Puppet Forge community members can also rate a module. Additional hyperlinks are provided to a list of outstanding issues listed by the developer, compatibility issues, and a page that explains how the module was scored/rated.

Searches can also be performed by the **puppet** command-line utility. The only requirement is that the search is performed on a Puppet host that is connected to the Internet. The following sample output shows the resulting output from a search on the term "firewall".

```
[root@host ~]# puppet module search firewall | head
Notice: Searching https://forgeapi.puppetlabs.com ...
NAME                          DESCRIPTION           AUTHOR           KEYWORDS
gildas-firewall               Unified Firewall ...  @gildas
puppetlabs-firewall           Manages Firewalls...  @puppetlabs      firewall
example42-firewall            Puppet firewall a...  @example42       firewall
sschneid-firewall             UNKNOWN               @sschneid        firewall
liamjbennett-windows_firewall  Module that will ...  @liamjbennett    firewall
camptocamp-firewall_c2c       Add autorequire t...  @camptocamp      firewall
thoward-windows_firewall      puppet windows fi...  @thoward
andrewkroh-base_firewall      Puppet module tha...  @andrewkroh
```

Like the Puppet Forge web interface, searches match modules based on their name, description, and keywords, or tags.

References

For more information, see
> Puppet Forge
> https://forge.puppetlabs.com

For more information, see
> Puppet Forge - Puppet Approved Modules
> https://forge.puppetlabs.com/approved

For more information, see
> Puppet Forge - Puppet Supported Modules
> https://forge.puppetlabs.com/supported

puppet-module(8) man page

Quiz: Puppet Modules In Puppet Forge

Choose the correct answer to the following questions.

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Puppet Forge includes links to documentation and various module development tools.

   a. True
   b. False

2. Which one of the following is a true statement?

   a. Puppet Labs supports approved modules published on Puppet Forge.
   b. Puppet Forge only publishes modules written by Puppet Labs.
   c. Puppet Forge is an environment where open source development of Puppet modules can occur.
   d. Approved modules do not have support of any kind.

3. Which one of the following requirements must be provided for the **`puppet module search`** command to work?

   a. A valid Puppet Forge username and password
   b. The full module name, including the author
   c. Internet access on the Puppet host
   d. The *Puppet Forge* [https://forge.puppetlabs.com] URL

4. What pieces of module information do Puppet Forge searches compare against? (Choose three.)

   a. Name
   b. Author
   c. Description
   d. License
   e. Source URL
   f. Tag

Solution

Choose the correct answer to the following questions.

When you have completed the quiz, click **check**. If you wish to try again, click **reset**. Click **show solution** to see all of the correct answers.

1. Puppet Forge includes links to documentation and various module development tools.

   a. True
   b. False

2. Which one of the following is a true statement?

   a. Puppet Labs supports approved modules published on Puppet Forge.
   b. Puppet Forge only publishes modules written by Puppet Labs.
   c. Puppet Forge is an environment where open source development of Puppet modules can occur.
   d. Approved modules do not have support of any kind.

3. Which one of the following requirements must be provided for the **`puppet module search`** command to work?

   a. A valid Puppet Forge username and password
   b. The full module name, including the author
   c. Internet access on the Puppet host
   d. The *Puppet Forge* [https://forge.puppetlabs.com] URL

4. What pieces of module information do Puppet Forge searches compare against? (Choose three.)

   a. Name
   b. Author
   c. Description
   d. License
   e. Source URL
   f. Tag

Implementing a Module from Puppet Forge

Objectives

After completing this section, students should be able to:

- Implement a Puppet module from Puppet Forge.

Although the Puppet Forge web interface is more pleasing to the eye, installing modules from Puppet Forge is better done from the command line. Each page that describes a Puppet module has a **download latest tar.gz** link at the bottom of the overview frame. This link can be used to download the module as a tar archive, then the **puppet module install *MODULE*.tar.gz** command installs the module, but not any of its dependencies. Dependencies have to be downloaded and installed first, or **puppet module install** can be used with the **--ignore-dependencies** option to force module installation.

When a full module name (including the author) is used with **puppet module install**, the command will download and install the module and its dependencies from Puppet Forge as a single command. The following sample output illustrates this with the **puppetlabs-firewall** module.

```
[root@host ~]# puppet module install puppetlabs-firewall
Notice: Preparing to install into /etc/puppet/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppet/modules
└── puppetlabs-firewall (v1.7.1)
```

Another useful option to **puppet module install** is the **--module_repository *REPOPATH*** option. It points the **puppet** command to a different Puppet module repository than Puppet Forge.

Note

Creating a private Puppet module repository is beyond the scope of this chapter.

References

puppet-module(8) man page

For more information, see

> Puppet Labs - Installing Modules
> http://docs.puppetlabs.com/puppet/3.6/reference/modules_installing.html

Guided Exercise: Implement a Puppet Module from Puppet Forge

In this lab, you will search Puppet Forge for a module that manages the message of the day. You will download and install the module, then apply its smoke test.

Resources	
Application URL	*Puppet Forge* [https://forge.puppetlabs.com/]
Machines	**servera**

Outcome(s)
You should be able to use Puppet Forge to search for, download, and install external Puppet modules.

Before you begin
The *puppet* package should already be installed on your **servera** host. Your host should have external Internet connectivity.

1. Log into **servera** as **root** and install the *tree* application. It will be used later to display the file hierarchy of the downloaded module.

```
[root@servera ~]# yum -y install tree
```

2. Search for a message of the day module ("motd") both online and from the command line using the **puppet module** command.

   2.1. Open a web browser and go to the *Puppet Forge* [https://forge.puppetlabs.com/] site. Enter **motd** in the search box, then click the **Find** button.

   2.2. Use the **puppet module search** command to search for modules that match the string, "motd". A few matches should display similar to the following output.

```
[root@servera ~]# puppet module search motd
Notice: Searching https://forgeapi.puppetlabs.com ...
NAME                      DESCRIPTION              AUTHOR            KEYWORDS
saz-motd                  Manage 'Message Of T...  @saz              rhel motd
attachmentgenie-motd      Puppet motd Module       @attachmentgenie  motd
CERNOps-motd              Manages entries in a...  @CERNOps          motd
fvoges-motd               Simple MOTD Puppet m...  @fvoges           motd unix
... Output omitted ...
```

3. See if Puppet Labs provides a module to manage the message of the day. Download and install it into **root**'s home directory.

```
[root@servera ~]# puppet module search motd | grep @puppetlabs
puppetlabs-motd           A simple module to d...  @puppetlabs       testing
```

The following example output shows what happens when the **puppetlabs-motd** module is installed. Using the **--modulepath=.** option causes Puppet to install the module below the current directory.

```
[root@servera ~]# puppet module install --modulepath=. puppetlabs-motd
Notice: Preparing to install into /root ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/root
└── puppetlabs-motd (v1.2.0)
```

puppet module install downloads the module and any dependencies from Puppet Forge, then installs them.

4. Run the **tree** command on the new directories that were created when modules were installed. The modules were saved in directories that included the module name without the leading author information.

```
[root@servera ~]# tree -d -L 1 motd
motd
├── manifests
├── spec
├── templates
└── tests

4 directories
```

5. Apply the **motd** class using the smoke test manifest provided in the **motd** module.

```
[root@servera ~]# puppet apply --modulepath=. motd/tests/init.pp
Warning: Config file /etc/puppet/hiera.yaml not found, using Hiera defaults
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 0.57 seconds
Notice: /Stage[main]/Motd/File[/etc/motd]/content: content changed
 '{md5}d41d8cd98f00b204e9800998ecf8427e' to '{md5}88e0d501d7e3aa61c2675f2ac47e59a4'
Notice: Finished catalog run in 0.14 seconds
```

The module updated the message of the day file, **/etc/motd**.

```
[root@servera ~]# cat /etc/motd
The operating system is RedHat
The free memory is 1.47 GB
The domain is lab.example.com
```

Lab: Implementing External Puppet Modules

In this lab, you will search Puppet Forge for the Puppet Labs module that provides the Apache web server. You will install the module and use it to build a web server.

Resources	
Files	**/var/www/html/index.html**
Application URL	*Puppet Forge* [https://forge.puppetlabs.com/]
Machines	**servera**

Outcome(s)
You should be able to use Puppet Forge to search for, download, and install external Puppet modules.

Before you begin
The *puppet* package should already be installed on your **servera** host. Your host should have external Internet connectivity.

1. Log into **workstation** as **student** and run the lab setup script. It cleans up previous lab work by erasing the *httpd* package and removing an existing **index.html** page.

```
[student@workstation ~]$ lab puppet-external setup
```

2. Search Puppet Forge and identify the Apache web server module that is published by Puppet Labs.

3. Install the Puppet Labs Apache module and any dependencies from Puppet Forge into the **/root** directory.

4. Create a smoke test file, **~/apache/tests/apache-test.pp**, which tests the **apache** module. Apply it and confirm it functions.

5. Open up the relevant firewall port.

6. Create a **/var/www/html/index.html** file that contains the following content:

```
The external Puppet lab is done.
```

7. Confirm the web content is being published.

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-external grade
```

Solution

In this lab, you will search Puppet Forge for the Puppet Labs module that provides the Apache web server. You will install the module and use it to build a web server.

Resources	
Files	**/var/www/html/index.html**
Application URL	*Puppet Forge* [https://forge.puppetlabs.com/]
Machines	**servera**

Outcome(s)
You should be able to use Puppet Forge to search for, download, and install external Puppet modules.

Before you begin
The *puppet* package should already be installed on your **servera** host. Your host should have external Internet connectivity.

1. Log into **workstation** as **student** and run the lab setup script. It cleans up previous lab work by erasing the *httpd* package and removing an existing **index.html** page.

```
[student@workstation ~]$ lab puppet-external setup
```

2. Search Puppet Forge and identify the Apache web server module that is published by Puppet Labs.

   Since Puppet Labs is the publisher, use **puppet module search** to see if there is a Puppet Labs "apache" module.

```
[root@servera ~]# puppet module search apache | grep @puppetlabs
Notice: Searching https://forgeapi.puppetlabs.com ...
NAME              DESCRIPTION                    AUTHOR        KEYWORDS
puppetlabs-apache  Installs, configures, and m...  @puppetlabs   web ssl rhel
```

   There is a module in Puppet Forge called **puppetlabs-apache** that will install and manage that server.

3. Install the Puppet Labs Apache module and any dependencies from Puppet Forge into the **/root** directory.

   Use the **puppet module install** command to perform this step. Adding the **--modulepath=.** option causes Puppet to install the module in the current directory.

```
[root@servera ~]# puppet module install --modulepath=. puppetlabs-apache
Notice: Preparing to install into /root ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/root
└─┬ puppetlabs-apache (v1.6.0)
  ├── puppetlabs-concat (v1.2.4)
  └── puppetlabs-stdlib (v4.9.0)
```

Note that the **puppetlabs-concat** and **puppetlabs-stdlib** modules are installed because **puppetlabs-apache** depends on them.

4. Create a smoke test file, **~/apache/tests/apache-test.pp**, which tests the **apache** module. Apply it and confirm it functions.

```
[root@servera ~]# mkdir apache/tests
[root@servera ~]# echo 'include apache' > apache/tests/apache-test.pp
[root@servera ~]# puppet apply --modulepath=. apache/tests/apache-test.pp
Warning: Config file /etc/puppet/hiera.yaml not found, using Hiera
 defaults
Notice: Compiled catalog for servera.lab.example.com in environment
 production in 3.14 seconds
... Output omitted ...
```

5. Open up the relevant firewall port.

   Default Apache behavior is to listen for HTTP connections on port 80. The following **firewall-cmd** commands will open that port immediately and persistently.

```
[root@servera ~]# firewall-cmd --add-service=http
success
[root@servera ~]# firewall-cmd --permanent --add-service=http
success
```

6. Create a **/var/www/html/index.html** file that contains the following content:

```
The external Puppet lab is done.
```

```
[root@servera ~]# cd /var/www/html
[root@servera html]# echo 'The external Puppet lab is done.' > index.html
```

7. Confirm the web content is being published.

   Open a browser and provide the following URL in the address frame: **http://servera.lab.example.com**.

```
[student@workstation ~]$ curl http://servera.lab.example.com
The external Puppet lab is done.
```

8. Run the lab grading script from **workstation** to check your work. The grading program will access **servera** to evaluate your work.

```
[student@workstation ~]$ lab puppet-external grade
```

Summary

In this chapter, you learned:

- *Puppet Forge* [https://forge.puppetlabs.com] has Puppet module resources for both developers and system administrators.
- Puppet Forge searches for modules based on their name, description, and tags assigned to them.
- The **`puppet module search`** command searches Puppet Forge for a module when the Internet is available.
- The **`puppet module install`** command will download a Puppet module and its dependencies from Puppet Forge, and install them, when the full module name is specified instead of a tar archive.
- The **`--ignore-dependencies`** option forces Puppet to install a module from a tar archive.

CHAPTER 13

IMPLEMENTING PUPPET IN A DEVOPS ENVIRONMENT

Overview	
Goal	Implement Puppet in a DevOps environment.
Objectives	• Describe Puppet in a DevOps environment and provision Vagrant machines. • Deploy Vagrant in a DevOps environment.
Sections	• Provisioning Vagrant Machines (and Guided Exercise) • Deploying Vagrant in a DevOps Environment (and Guided Exercise)
Lab	• Implementing Puppet in a DevOps Environment

Provisioning Vagrant Machines

Objectives

After completing this section, students should be able to:

- Describe Puppet in a DevOps environment and provision Vagrant machines.

DevOps in the enterprise

Development and operations staff both bear the responsibility of leveraging technology to fulfill the goals of the business, whether it be creating applications to streamline internal business processes and workflows or producing innovative products and solutions for customers. While their responsibilities may appear aligned, the two groups often differ completely in their objectives and methodologies.

Developers prefer to work fast and without boundaries since change is a requirement for innovation. They desire an agile work environment without obstacles that could hinder the rapid iterations of development cycles needed to keep up with ever-changing business requirements or the never-ending list of new product features needed to remain competitive in the marketplace. Therefore, developers often demand **`root`** account access to avoid running into privilege issues, and full operating system installs so they will not encounter missing software dependencies.

On the other hand, operations groups are primarily concerned with providing a stable and reliable infrastructure for mission-critical applications to run on. They also have to ensure adherence to security policies and regulatory compliance. Since change is the enemy of stability, they try to create an environment that is as consistent and as static as possible. In order to maintain the integrity of the infrastructure, they are extremely conservative regarding the frequency and magnitude of changes that are allowed into their environment.

The conflicting philosophies and practices of developers and system administrators often come to a head when the two teams come together during the deployment of new software releases to production environments. Due to the disparities between the unrestricted development environment and strictly controlled production environment, it is no surprise that stressful firefighting conference calls and lengthy post-mortem meetings routinely follow production deployments.

In recent years, the DevOps approach has seen increasing adoption in the enterprise as organizations strive to resolve the conflict between their development and operational teams. DevOps' namesake is derived from its core principle that software development and operations performance can be improved and accelerated through better communication, integration, and cooperation between software developers and IT operations professionals.

Infrastructure as code

DevOps places a strong focus on the ability to build and maintain essential components with automated, programmatic procedures. One key DevOps concept is the idea of *infrastructure as code*. This concept is an important and groundbreaking paradigm shift for the many system administrators who currently manage their infrastructure through manual execution of administrative commands and editing of configuration files. By designing, implementing, and managing infrastructure as code, configurations can be predictably and consistently deployed and replicated throughout an environment.

In recent years, the Puppet software has become a popular tool for managing infrastructure code. Puppet allows tasks such as software installations and server configurations to be automated. This automation removes the introduction of human errors resulting from manual administration and therefore greatly improves operational efficiency while simultaneously enhancing the predictability of successful outcomes.

Puppet's master/client architecture provides for centralization of configuration changes. This design results in greater control over the infrastructure code by effectively implementing a single point of entry for changes to the infrastructure.

Puppet's declarative nature also makes it self-documenting and helps administrators easily get a clear picture of their infrastructure configuration. In addition to enforcing configuration end states, Puppet also provides administrators the ability to audit system configurations and detect when they deviate from the desired state.

While configuration management tools like Puppet simplify and improve infrastructure management, they also introduce the dilemma of how this new infrastructure code should be managed. To overcome this challenge, administrators would do well to take a page out of the books of their development counterparts. In accordance with development best practices, administrators should use a version control system, such as Git, to manage their infrastructure code.

Version control allows administrators to implement a life cycle for the different stages of their infrastructure code, such as development, QA, and production. By managing infrastructure code with a version control tool, administrators can test their infrastructure code changes in noncritical development and QA environments to minimize surprises and disruptions when deployments are implemented in production environments.

Vagrant

When testing code for production deployment, results are only relevant and valid if a test is conducted in a development environment which is identical to the production environment. This is as true for software development as it is for the changes to infrastructure code. Virtualization technology makes it easy and cost-efficient to stand up a machine for testing code before production deployment. However, the real challenge is how to construct a development environment on the virtual machine so that it is a replica of the production environment.

Vagrant is a useful tool for overcoming this challenge. While the practice of managing infrastructure as code makes it possible to ensure identical system and software configurations exist between development and production environments, Vagrant further simplifies the process of creating and managing consistent virtualized environments needed for development.

Following the infrastructure-as-code approach, Vagrant automates the creation of a virtual machine, its hardware configuration, software installation, system configuration, and development source code retrieval by allowing the entire process to be specified within a plain text configuration file. With Vagrant, deploying a development environment can be as simple as checking out a project from version control and executing **`vagrant up`** on the command line.

An additional benefit of Vagrant's management of virtualized environments as code is that it is very easy not just to create a virtualized development environment, but also to share the identical environment amongst different team members. Since it insulates the end user from the complexities of setting up and sharing identical virtualized development environments, Vagrant is an ideal tool not only for administrators to test infrastructure code changes but also for developers to test software releases.

Vagrant components
The Vagrant software is composed of the following main components:

Box: A box is a **tar** archive which contains a virtual machine image. A box file serves as the foundation of a Vagrant virtualized environment and is used to create virtual machine instances. For greater flexibility, the image should contain just a base operating system installation. This allows the image to be used as a starting point for creating different virtual machines regardless of the specific requirements of their applications. These application-specific requirements can be fulfilled through automated configuration after the virtual machine is created.

Provider: A provider allows Vagrant to interface with the underlying platform that a Vagrant box image is deployed on. Vagrant comes packaged with a provider for Oracle's VirtualBox. Alternative providers are currently available for other virtualization platforms such as VMware, Hyper-V, and KVM.

Vagrantfile: Vagrantfile is a plain text file that contains the instructions for creating a Vagrant virtualized environment. The instructions are written using Ruby syntax. The contents of this file can be used to prescribe how the virtual machine is to be built and configured.

Note

Administrators can create their own Vagrant box images or leverage existing images publicly available at HashiCorp's Atlas box catalog located at **http://www.vagrantcloud.com**. This course will utilize a premade Red Hat Enterprise Linux box image since the creation of a Vagrant box image is beyond the scope of this course.

Important

Box files and their contained images are specific to each provider. For example, a box file created for use with the VirtualBox provider is not compatible with the VMware provider. Organizations using multiple providers will need to have separate box files that are specific to each provider.

Configuring a Vagrant environment

Vagrant environments are meant to be operated in isolation from each other. To create a new Vagrant environment, start by creating a project directory for the new environment. Within this project directory, create a **Vagrantfile** containing the instructions for deploying a new Vagrant machine. The following example shows how developers can initiate Vagrant projects on their workstations.

```
[root@host ~]# mkdir -p /root/vagrant/project
[root@host ~]# cd /root/vagrant/project
[root@host project]# vi Vagrantfile
```

Creating a basic Vagrantfile
The following lines from a **Vagrantfile** file provide instructions to Vagrant for the creation of a basic virtual machine. The **config.vm.box** method call specifies that the virtual machine be cloned from a box image named **rhel7.1**, which can be obtained from the location specified

by the **config.vm.box_url** method call. The **config.vm.hostname** method call instructs Vagrant to name the virtual machine **sandbox.example.com** once it is created.

```
Vagrant.configure(2) do |config|
  config.vm.box = "rhel7.1"
  config.vm.box_url = "http://content.example.com/puppet3.6/x86_64/dvd/vagrant/rhel-
server-libvirt-7.1-1.x86_64.box"
  config.vm.hostname = "sandbox.example.com"
end
```

Managing Vagrant machines

With a **Vagrantfile** file created, the Vagrant machine can be instantiated by executing the **vagrant up** command from the root of the project directory.

```
[root@host project]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
==> default: Box 'rhel7.1' could not be found. Attempting to find and install...
    default: Box Provider: libvirt
... Output omitted ...
==> default: Waiting for SSH to become available...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
... Output omitted ...
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Rsyncing folder: /root/vagrant/project/ => /home/vagrant/sync
```

The output of **vagrant up** showcases all the configuration and administration tasks that Vagrant automates and insulates the user from. These tasks include retrieval of the box image, creation of the virtual machine, setting the host name, configuring network interfaces, and copying files from the host to the Vagrant machine. If static IP addressing is not defined in **Vagrantfile**, the virtual machine will be dynamically assigned an IP address by the virtualization provider.

Once the Vagrant machine has started, it can be accessed using the **vagrant ssh** command. During the deployment of the Vagrant machine, an SSH key is generated on the host and then installed in the **~/.ssh** directory of the **vagrant** user on the Vagrant machine. The **vagrant ssh** command uses this key to authenticate an SSH session to the Vagrant machine as the **vagrant** user.

```
[root@host project]# vagrant ssh
Last login: Wed Oct 21 14:02:44 2015 from 192.168.121.1

[vagrant@sandbox ~]$ id
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant),1001(docker)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Note

The Vagrant software expects a **vagrant** user account to be available on the Vagrant machine for SSH access. Therefore, it is standard practice to create the **vagrant** user account and configure its sudo privileges during the creation of base virtual machine images. While SSH key-based authentication is used by Vagrant, it is also standard practice to set the password of the **vagrant** user account to '**vagrant**'.

Vagrant also offers a helpful feature called *synced folders*. By default, Vagrant uses the synced folder feature to copy the content of the project directory to a directory on the Vagrant machine (**~/sync/**) that is accessible by the **vagrant** user.

```
[root@host project]# ls -l
total 4
-rw-r--r--. 1 root root 227 Oct 21 13:50 Vagrantfile
```

```
[vagrant@sandbox project]$ ls -l /home/vagrant/sync
total 4
-rw-r--r--. 1 vagrant vagrant 227 Oct 21 13:50 Vagrantfile
```

To allow for the execution of privileged commands on Vagrant machines, box images are built with **sudo** privileges granted to the **vagrant** user. This **sudo** privilege proves especially useful when configuring more advanced Vagrant machine deployments, which will be discussed later.

```
[vagrant@sandbox project]$ sudo -l
Matching Defaults entries for vagrant on this host:
    !visiblepw, always_set_home, env_reset, env_keep="COLORS DISPLAY HOSTNAME
    HISTSIZE INPUTRC KDEDIR LS_COLORS", env_keep+="MAIL PS1 PS2 QTDIR USERNAME
    LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT
    LC_MESSAGES", env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE",
    env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
    secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin

User vagrant may run the following commands on this host:
    (ALL) NOPASSWD: ALL
```

When a Vagrant machine is no longer needed, execute the **vagrant halt** command to shut down the Vagrant machine. Another option is to execute the **vagrant destroy** command to stop the running machine, as well as clean up all the resources which were created when the machine was deployed.

```
[vagrant@sandbox project]$ exit
logout
Connection to 192.168.121.85 closed.

[root@host project]# vagrant destroy
==> default: Removing domain...
```

Vagrant provisioners

The previous example demonstrated the use of Vagrant for quick deployment of a simple virtual machine. While this is convenient, the real strength of Vagrant for creating development systems for DevOps environments lies in its provisioning feature.

As mentioned previously, a box image should contain just the base operating system so the image can serve as the foundation for the customization of different virtual machines. Vagrant's provisioning feature automates the software installation and configuration changes needed to overlay the customizations over the base operating system.

Provisioning is performed with the use of one or more *provisioners* offered by Vagrant. Provisioners are enabled in a **`Vagrantfile`** file with the use of the **`config.vm.provision`** method call. The following example demonstrates how Vagrant's shell provisioner can be used to automate the configuration of yum repositories and the installation of the Puppet software after the Vagrant machine is deployed with a base operating system.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.vm.provision "shell", inline: <<-SHELL
    sudo cp /home/vagrant/sync/etc/yum.repos.d/* /etc/yum.repos.d
    sudo yum install -y puppet
  SHELL
end
```

References

Vagrant Boxes
https://docs.vagrantup.com/v2/boxes.html

Vagrant Providers
https://docs.vagrantup.com/v2/providers/index.html

Vagrantfile
https://docs.vagrantup.com/v2/vagrantfile/index.html

Vagrant Command-Line Interface
https://docs.vagrantup.com/v2/cli/index.html

Vagrant Provisioning
https://docs.vagrantup.com/v2/provisioning/index.html

Guided Exercise: Provisioning Vagrant Machines

In this lab, you will provision a Vagrant machine with Puppet for use in a DevOps environment.

Resources	
Files	**/root/vagrant/webapp/Vagrantfile**, **/root/vagrant/webapp/etc/puppet/yum.repos.d/rhel_dvd.repo**, and **/root/vagrant/webapp/etc/puppet/yum.repos.d/rhsat_dvd.repo**
Application URL	**http://content.example.com/puppet3.6/x86_64/dvd/vagrant/rhel-server-libvirt-7.1-1.x86_64.box**, **http://content.example.com/puppet3.6/x86_64/dvd/vagrant/Vagrantfile**
Machines	**serverb**

Outcome(s)
You should be able to deploy a Vagrant machine with *puppet* installed.

Before you begin
Reset your **serverb** server.

1. Log into **workstation** as **student** and run the **puppet-devops-practice** lab setup script. The script will make the Vagrant software available on **serverb**.

```
[student@workstation ~]$ lab puppet-devops-practice setup
```

2. You will use Git to manage changes to your Vagrant configuration. As **student** on **workstation**, create a central repository.

```
[student@workstation ~]$ git init --bare --shared=true /var/git/vagrant.git
Initialized empty shared Git repository in /var/git/vagrant.git/
```

3. Log into **serverb** as the **root** user and install and configure Git. Download the **config-git**

```
[root@serverb ~]# curl -O http://materials.example.com/config-git
... Output omitted ...
[root@serverb ~]# chmod 755 config-git
[root@serverb ~]# ./config-git
Install the Git software .....................
... Output omitted ...
Define Git configurations .....................
Deploying SSH keys for authentication .........
The authenticity of host 'workstation (172.25.250.254)' can't be established.
ECDSA key fingerprint is c3:8d:dd:ce:4f:df:f6:fc:40:9b:ee:de:14:ae:7f:30.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
 any that are already installed
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
 now it is to install the new keys
student@workstation's password: student

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'student@workstation'"
and check to make sure that only the key(s) you wanted were added.
```

config-git configures SSH key authentication to the **student** account on **workstation**. Use **ssh** to confirm the keys are configured properly.

```
[root@serverb ~]# ssh student@workstation
Last login: Wed Nov  4 22:36:03 2015 from 172.25.252.254
```

4. Create a work directory, **/root/vagrant**, for Vagrant work. Clone it from the central **vagrant** Git repository. Also create a subdirectory, **webapp**, inside the Vagrant work directory.

```
[root@serverb ~]# git clone ssh://student@workstation/var/git/vagrant.git
Cloning into 'vagrant'...
warning: You appear to have cloned an empty repository.
[root@serverb ~]# mkdir vagrant/webapp
```

5. In the **/root/vagrant/webapp** directory, create a Vagrant configuration file, **Vagrantfile** by downloading it from **http://materials.example.com/vagrant/Vagrantfile**. This file will create a Vagrant machine using the box image provided in the previous table, name the machine box image **rhel7.1**, and configure the machine with a host name of **dev.lab.example.com**.

```
[root@serverb ~]# cd vagrant/webapp
[root@serverb webapp]# curl -O http://materials.example.com/vagrant/Vagrantfile
```

6. Put the **webapp** subdirectory and **Vagrantfile** under Git control. Check them into the local repository.

```
[root@serverb webapp]# git add .
[root@serverb webapp]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   Vagrantfile
#
[root@serverb webapp]# git commit -m 'Initial version.'
... Output omitted ...
```

7. Test the deployment of the virtual machine with the new configuration.

```
[root@serverb webapp]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
```

```
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings...
... Output omitted ...
```

8. Verify that the *puppet* package is *not* installed in the default Red Hat Enterprise Linux 7.1 box. Also check to see which Yum repositories are available in the box.

```
[root@serverb webapp]# vagrant ssh
[vagrant@dev ~]$ rpm -q puppet
package puppet is not installed
[vagrant@dev ~]$ yum repolist
Loaded plugins: product-id, subscription-manager
repolist: 0
```

9. Since the *puppet* package is not installed and no Yum repositories are available, configure Vagrant so that the necessary Yum repository configuration files are created when the Vagrant machine is deployed.

   9.1. Exit from and shut down the Vagrant machine. Note that your IP address may be different than the one shown in the example.

```
[vagrant@dev ~]$ exit
logout
Connection to 192.168.121.238 closed.

[root@serverb webapp]# vagrant destroy
==> default: Removing domain...
```

   9.2. Create a directory within the Vagrant work directory to host the Yum repository configuration files and then change to that directory.

```
[root@serverb webapp]# mkdir -p /root/vagrant/webapp/etc/yum.repos.d
[root@serverb webapp]# cd /root/vagrant/webapp/etc/yum.repos.d
```

   9.3. Create a Yum repository configuration file called **`rhel_dvd.repo`** with the following contents by copying it from **`/etc/yum.repos.d/rhel_dvd.repo`**:

```
[rhel_dvd]
gpgcheck = 0
enabled = 1
baseurl = http://content.example.com/rhel7.1/x86_64/dvd
name = Remote classroom copy of dvd
```

   9.4. Create a Yum repository configuration file called **`rhsat_dvd.repo`** with the following contents by copying it from **`/etc/yum.repos.d/rhsat_dvd.repo`**:

```
[rhsat_dvd]
gpgcheck = 0
enabled = 1
baseurl = http://content.example.com/rhsat6.1/x86_64/dvd
name = Remote classroom copy of RH Satellite dvd
```

   9.5. Change back to the **webapp** directory.

```
[root@serverb webapp]# cd /root/vagrant/webapp
```

9.6. Modify the Vagrant configuration file so that Puppet is installed during provisioning. The Yum repository configuration files on the host must be copied to **/etc/yum.repos.d** in the Vagrant machine. Once Yum is configured, the *puppet* package can be installed.

Add the following lines inside the **Vagrant.configure** code block in the **Vagrantfile** file. They should be inserted immediately below the "# Define shell provisioner" comment.

```
config.vm.provision "shell", inline: <<-SHELL
  sudo cp /home/vagrant/sync/etc/yum.repos.d/* /etc/yum.repos.d
  sudo yum -y install puppet
SHELL
```

10. Provision the Vagrant machine using the newly modified configuration.

```
[root@serverb webapp]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings...
... Output omitted ...
==> default:   rubygems.noarch 0:2.0.14-24.el7
==> default:
==> default: Complete!
```

11. Verify that Yum repositories are now available and that the *puppet* package is installed on the Vagrant machine.

```
[root@serverb webapp]# vagrant ssh

[vagrant@dev ~]$ yum repolist
Loaded plugins: product-id, subscription-manager
repo id      repo name                                    status
rhel_dvd     Remote classroom copy of dvd                 4,371
rhsat_dvd    Remote classroom copy of RH Satellite dvd     408
repolist: 4,779
[vagrant@dev ~]$ rpm -q puppet
puppet-3.6.2-4.el7sat.noarch
```

12. Exit and shut down the Vagrant machine.

```
[vagrant@dev ~]$ exit
[root@serverb webapp]# vagrant destroy
```

13. Save your changes with a descriptive log message in the central Git repository.

13.1. List all of the files in the **webapp** directory.

```
[root@serverb webapp]# ls -a
.  ..  etc  .vagrant  Vagrantfile
```

13.2. The **.vagrant** subdirectory is where Vagrant stores its metadata. Prevent this directory from being controlled by Git by creating a **.gitignore** file that includes its name.

```
[root@serverb webapp]# echo .vagrant > .gitignore
```

13.3. Add the files in the current directory to the staging area, then commit them to the local repository with a meaningful log message.

```
[root@serverb webapp]# git add .
[root@serverb webapp]# git commit -m 'Configured Yum and installed Puppet.'
... Output omitted ...
```

13.4. Push all of the changes to the upstream Git repository.

```
[root@serverb webapp]# git push
... Output omitted ...
```

Deploying Vagrant In a DevOps Environment

Objectives

After completing this section, students should be able to:

- Deploy Vagrant in a DevOps environment.

Integrating Vagrant with Puppet

In the previous section, deploying a Vagrant machine with a base operating system was shown. Vagrant's provisioning feature and the shell provisioner were shown to install software, such as Puppet, after the Vagrant machine was created from the base operating system image. The installation of Puppet on the Vagrant machine allows the machine to be integrated into an organization's Puppet infrastructure.

Vagrant offers a **`puppet_server`** provisioner that facilitates the software installation and system configuration of a Vagrant machine using an organization's Puppet infrastructure. This provisioner can automate the configuration of the Puppet agent, the connection to the organization's Puppet master, and the retrieval of the applicable Puppet modules and manifests.

When using Vagrant's **`puppet_server`** provisioner in DevOps environments, it is entirely possible to create a Vagrant machine that is configured identically to production systems. In both cases, the systems are seeded with base operating system installations and then configured when modules and manifests are applied from the Puppet master. As long as the same modules and manifest applied to a production system are applied to a Vagrant machine, both systems can be nearly identical. The Vagrant machine will then serve as a valid and useful development environment for testing software release or infrastructure code changes before production deployment.

Using synced folders

Vagrant offers several other features to facilitate the provisioning process. As mentioned previously, one helpful feature is synced folders. By default, Vagrant uses the feature to copy contents of the project directory on the host to a directory on the Vagrant machine during its instantiation.

Additional content can be copied from the host to the Vagrant machine by configuring additional synced folders in the **`Vagrantfile`** configuration file. Vagrant offers various mechanisms for keeping these folders in sync. The following example shows how to add to **`Vagrantfile`** a synced folder that uses **`rsync`** for content synchronization. Vagrant's default synced folder type is **`VirtualBox`**. This folder type offers a bidirectional sync of the contents in the host and Vagrant machine folder, and will continuously sync contents as changes are introduced to folder contents on either the host or the Vagrant machine.

However, **`VirtualBox`** synced folders are only available when using the **`VirtualBox`** provider. When using another provider, a different sync folder type will need to be used. Be sure to consult the documentation to understand how each synced folder type works, since they can behave differently.

Since this course does not use the **`VirtualBox`** provider, the **`rsync`** synced folder type will be used instead. Unlike **`VirtualBox`** synced folders, **`rsync`** synced folders are unidirectional and will only copy files from the host to the Vagrant machine and not the other way around. In addition, the **`rsync`** mechanism does not constantly copy changes to folder contents. It will

only copy folder contents from the host to the Vagrant machine when **vagrant up** is executed. If folder contents need to be recopied after this initial synchronization, execute the **vagrant rsync** command as shown in the following example.

```
[root@host project]# vagrant rsync
==> default: Rsyncing folder: /root/vagrant/project/ => /home/vagrant/sync
```

The following example demonstrates how the **Vagrantfile** configuration file can be modified to add a synced folder which uses the **rsync** mechanism to copy the contents of the folder. As dictated by the **owner** and **group** options, the destination files will have both owner and group ownership set to **puppet**. In addition, since the **rsync** operation is performed using the **vagrant** account on the Vagrant machine, insufficient privileges will cause the file copying to fail. The option **rsync__rsync_path** is used to specify that the **sudo rsync** command should be used on the Vagrant machine instead of the default **rsync** command.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.vm.synced_folder "puppet/ssl", "/var/lib/puppet/ssl", type: "rsync", owner:
 "puppet", group: "puppet", rsync__rsync_path: "sudo rsync"
end
```

Configuring Vagrant for Puppet provisioning

The following procedure illustrates how to configure Vagrant to use the **puppet_server** provisioner to automate software and configuration deployments to a Vagrant machine. The following example shows how to deploy a Vagrant machine on **sandbox.example.com** and configure it to communicate with the Puppet master running on **puppetmaster.example.com**.

1. Modify the Vagrant configuration file so that the Vagrant machine communicates with the Puppet master running on **puppetmaster.example.com** when launched.

   1.1. Add the following lines inside the **Vagrant.configure** code block in the **Vagrantfile** file. The **puppet.options** parameter specifies the Puppet client options necessary for certificate signing to take place upon first time connection to the Puppet master. It also specifies for the system to retrieve and apply its configuration from the Puppet master.

   ```
   Vagrant.configure(2) do |config|

     ... Configuration omitted ...

       config.vm.provision "puppet_server" do |puppet|
         puppet.puppet_server = 'puppetmaster.example.com'
         puppet.options = '--onetime --verbose --waitforcert=10 --no-
   usecacheonfailure --no-daemonize'
       end
     end
   ```

   1.2. Launch the Vagrant machine. It should successfully register with the Puppet master and retrieve its configuration. The configuration applied will enable and start the **httpd** service on **sandbox.example.com**.

   ```
   [root@sandbox project]# vagrant up
   ```

```
Bringing machine 'default' up with 'libvirt' provider...
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings..
... Output omitted ...
==> default: Running provisioner: puppet_server...
==> default: Running Puppet agent...
==> default: Info: Creating a new SSL key for sandbox.example.com
==> default: Info: Caching certificate for ca
==> default: Info: csr_attributes file loading from /etc/puppet/
csr_attributes.yaml
==> default: Info: Creating a new SSL certificate request for
 sandbox.example.com
==> default: Info: Certificate Request fingerprint (SHA256):
 8B:67:38:82:F7:97:6B:25
==> default: Info: Caching certificate for sandbox.example.com
==> default: Info: Caching certificate_revocation_list for ca
==> default: Info: Caching certificate for ca
==> default: Info: Retrieving plugin
==> default: Info: Caching catalog for sandbox.example.com
==> default: Info: Applying configuration version '1444926117'
==> default: Notice: /Stage[main]/Main/Node[sandbox.example.com]/Package[httpd]/
ensure:
 created
==> default: Notice: /Stage[main]/Main/Node[sandbox.example.com]/Service[httpd]/
ensure:
 ensure changed 'stopped' to 'running'
==> default: Info: /Stage[main]/Main/Node[sandbox.example.com]/Service[httpd]:
 Unscheduling refresh on Service[httpd]
==> default: Info: Creating state file /var/lib/puppet/state/state.yaml
==> default: Notice: Finished catalog run in 28.26 seconds
```

2. During the first run of the Puppet client, it will cache its new keys and certificates under the **/var/lib/puppet/ssl** directory on the Vagrant machine. These files need to be archived and made available to future Vagrant machine instances. If they are absent, future Vagrant machine instances generated for this project will attempt to generate a new keys and Puppet client certificate, which will result in failed communication with the Puppet master.

   Archive the certificates to the **puppet/ssl** subdirectory within the project directory and shut down the Vagrant machine.

   2.1. Create a directory on the host to archive the Puppet certificates.

   ```
   [root@sandbox project]# mkdir -p puppet/ssl
   ```

   2.2. Display the network and SSH information for the Vagrant machine. This will provide the IP address of the machine.

   ```
   [root@sandbox project]# vagrant ssh-config
   Host default
     HostName 192.168.121.88
     User vagrant
     Port 22
     UserKnownHostsFile /dev/null
     StrictHostKeyChecking no
     PasswordAuthentication no
     IdentityFile /root/vagrant/project/.vagrant/machines/default/libvirt/
   private_key
     IdentitiesOnly yes
   ```

```
  LogLevel FATAL
```

2.3. Copy the contents of the **/var/puppet/ssl** directory on the Vagrant machine to the archive directory on the host. Be sure to substitute the Vagrant machine's IP address in the command with the one displayed by the **vagrant ssh-config** command. The **--rsync-path** option to the **rsync** command will allow the file copy to be performed with sudo privilege on the Vagrant machine. The **-e** option to the **rsync** command specifies that the command will the performed through an SSH connection established with key-based authentication.

```
[root@sandbox project]# rsync -avz --rsync-path='sudo rsync' -e 'ssh -
i /root/vagrant/project/.vagrant/machines/default/libvirt/private_key'
 vagrant@192.168.121.88:/var/lib/puppet/ssl/ puppet/ssl/
The authenticity of host '192.168.121.88 (192.168.121.88)' can't be established.
ECDSA key fingerprint is 51:f7:72:0d:2d:e2:13:1b:84:19:61:41:6a:e5:d4:ac.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.121.88' (ECDSA) to the list of known hosts.
receiving incremental file list
./
crl.pem
certificate_requests/
certificate_requests/sandbox.example.com.pem
certs/
certs/ca.pem
certs/sandbox.example.com.pem
private/
private_keys/
private_keys/sandbox.example.com.pem
public_keys/
public_keys/sandbox.example.com.pem

sent 148 bytes  received 8723 bytes  2534.57 bytes/sec
total size is 10624  speedup is 1.20
```

2.4. Shut down the Vagrant machine.

```
[root@sandbox project]# vagrant destroy
```

3. Modify the Vagrant configuration file so that the contents of the **/root/vagrant/project/puppet/ssl** directory are synced to the **/var/lib/puppet/ssl** of the Vagrant machine when it is launched. Verify that the Puppet agent is able to communicate with the Puppet master.

3.1. Add the following line within the **Vagrant.configure** code block in the **Vagrantfile** file. This utilizes Vagrant's sync folder feature to copy the files from the host to the Vagrant machine.

```
Vagrant.configure(2) do |config|
   ... Configuration omitted ...

 config.vm.synced_folder "puppet/ssl", "/var/lib/puppet/ssl", type: "rsync",
 owner: "puppet", group: "puppet", rsync__rsync_path: "sudo rsync"
```

3.2. Verify the new configuration by launching a new Vagrant machine. The new machine should communicate successfully with the Puppet master using the keys and certificates archived from the initial Puppet client run.

```
[root@sandbox project]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings...
... Output omitted ...
==> default: Rsyncing folder: /root/vagrant/webapp/puppet/ssl/ => /var/lib/
puppet/ssl
... Output omitted ...
==> default: Running provisioner: puppet_server...
==> default: Running Puppet agent...
==> default: Info: Retrieving plugin
==> default: Info: Caching catalog for sandbox.example.com
==> default: Info: Applying configuration version '1444926117'
==> default: Notice: /Stage[main]/Main/Node[sandbox.example.com]/Package[httpd]/
ensure: created
==> default: Notice: /Stage[main]/Main/Node[sandbox.example.com]/Service[httpd]/
ensure: ensure changed 'stopped' to 'running'
==> default: Info: /Stage[main]/Main/Node[sandbox.example.com]/Service[httpd]:
 Unscheduling refresh on Service[httpd]
==> default: Info: Creating state file /var/lib/puppet/state/state.yaml
==> default: Notice: Finished catalog run in 28.20 seconds
```

Creating a Vagrant development environment

Once a Vagrant machine has been integrated and configured by an organization's Puppet infrastructure, it should have all the software needed for its intended purpose. For example, one would expect Puppet to install Apache and start web service on a Vagrant machine that is to be used to test a web application.

In addition to configuration management, the creation of a development environment can be automated further by incorporating application source code into the Vagrant project. Application source code can be incorporated by making it part of the Vagrant project directory. Using Vagrant's synced folder feature, the source code can then be copied from the host system into the Vagrant machine upon launch.

The following example shows how the contents of a web application's **`DocumentRoot`** folder can be incorporated into a **`www`** subdirectory in the Vagrant project directory. The contents of this folder will be copied to the **`/var/www/`** directory on the Vagrant machine when it is launched.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.vm.synced_folder "www", "/var/www/html", type: "rsync",  owner: "apache",
 group: "apache", rsync__rsync_path: "sudo rsync"
end
```

Using forwarded ports

If the application being developed has network services, Vagrant provides a networking configuration feature called *forwarded ports*, which maps network ports on the host system to ports on the Vagrant machine. The following example shows how the **`Vagrantfile`** file can be

modified so that Vagrant forwards traffic directed at port 8000 on the host system to port 80 on the Vagrant machine.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.vm.network :forwarded_port, guest: 80, host: 8000
end
```

Note

Configuration changes made to **Vagrantfile** will not have any effect on an existing running Vagrant machine. Changes will take effect when the machine is halted with **vagrant halt** or **vagrant destroy**, and then relaunched with **vagrant up**. An alternative is to execute the **vagrant reload** command. This command performs the same actions as **vagrant halt** followed by **vagrant up**. Provisioners defined with the **Vagrantfile** file will not be reexecuted when **vagrant reload** is issued.

Creating a reusable Vagrant development environment

Once a Vagrant project is configured for integration into an organization's Puppet infrastructure and has application source code incorporated into the project directory, it can be made into a reusable development environment that is easily deployed. Version control systems are widely used by developers to manage application source code. As mentioned previously, one of the advantages of Vagrant's design is that it follows the infrastructure-as-code approach. Since the configuration of a Vagrant machine is maintained as code, it too can also be managed by a version control system. By placing an entire Vagrant project directory in a version control system, such as Git, administrators effectively bundle all the components needed to recreate a ready-to-go Vagrant development environment together into a single Git project.

Suppose a new team of developers were tasked to work on the application source code. These developers merely need to install the Vagrant software on their workstations and then run **git clone** followed by **vagrant up**. The **git clone** command retrieves all the Vagrant project components (Vagrant configuration, application source code, etc.) while **vagrant up** recreates the Vagrant development environment using the retrieved components.

Since the recreation of the development environment is dictated by coded instructions, each developer ends up with a development environment that is not only identical to that on each of their teammates' workstations, but also identical to that on the production server. This provides the developers assurance that application source code changes validated on the Vagrant development environments on their workstations will behave identically when deployed to production. Perhaps the most elegant part of this design is that no work was required on the part of the operations staff to get these developers up and running.

The intelligence of this design becomes even more evident when changes are required by the operations team. As operations staff makes changes to the production environment, such as deploying new software versions to address bugs or security vulnerabilities, the same changes need to be made to the associated development environments to keep them identical. This is simply accomplished by making the production Puppet configurations available to the Vagrant development machines. The Puppet client on the development Vagrant machines applies the new configuration behind the scenes. The developers remain blissfully ignorant of this and their work is uninterrupted by these operational changes.

A development environment implemented in this manner with Vagrant allows both development and operations teams to operate in cooperation rather than in conflict with each other. Developers are empowered to easily deploy development environments that are replicas of the production environment by themselves without burdening the operations team. Since the operations team have control over development environments' resemblance to the production environment, code deployments to production should be uneventful.

Deploying code with `vagrant push`

Once code changes are successfully verified on a Vagrant development environment, there are a multitude of ways to propagate the new code to other downstream environments such as QA and production. Vagrant offers a feature for deploying application code from the Vagrant project directory to a specified destination. This feature is enabled by defining deployment routines in the project's **`Vagrantfile`**.

The following example shows how the **`Vagrantfile`** file can be modified to define a deployment routine that uses the **`rsync`** to copy web application code from the Vagrant project directory on **`host`** to the **`/var/www/html`** directory on **`web`**. In addition, the deployment routine also runs the **`restorecon`** on the remote destination directory to ensure that the appropriate SELinux context is set.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.push.define "local-exec" do |push|
    push.inline = <<-SCRIPT
      rsync -avz www/ web:/var/www/html/
      ssh web 'restorecon -rv /var/www/html'
    SCRIPT
  end
end
```

To invoke the deployment routine, issue the **`vagrant push`** command.

References

Vagrant Puppet Agent Provisioner
https://docs.vagrantup.com/v2/provisioning/puppet_agent.html

Vagrant Rsync Synced Folder
https://docs.vagrantup.com/v2/synced-folders/rsync.html

Vagrant Forwarded Ports
https://docs.vagrantup.com/v2/networking/forwarded_ports.html

Vagrant Push
https://docs.vagrantup.com/v2/push

Guided Exercise: Deploying Vagrant in a DevOps Environment

In this lab, you will create a development environment using Vagrant and Puppet.

Resources	
Files	**/root/vagrant/webapp/puppet/ssl**
Application URL	**serverc.lab.example.com:/var/git/webapp.git**
Machines	**serverb**, **serverc**

Outcome(s)
You should be able to create a reusable Vagrant development environment that has its configuration managed by Puppet.

Before you begin
serverb should have a working Vagrant machine configured with Puppet per a previous exercise.

Reset your **serverc** server so it can be configured with a Git repository.

1. Log into **workstation** and run the lab setup script to set up a Git repository on **serverc**.

```
[student@workstation ~]$ lab puppet-vagrant-practice setup
```

2. On **serverb**, modify the Vagrant configuration file so that the Vagrant machine registers with the Puppet master running on **serverc.lab.example.com** when launched. Launch the Vagrant machine and verify that the Puppet configuration is applied.

   2.1. Change directory to the **/root/vagrant/webapp** Vagrant project directory.

```
[root@serverb ~]# cd /root/vagrant/webapp
```

   2.2. Add the following lines inside the **Vagrant.configure** code block in **/root/vagrant/webapp/Vagrantfile**. The entry specifies the Puppet master and the options to use with the Puppet agent. The lines should be inserted immediately below the "# Define puppet_server provisioner" comment.

```
config.vm.provision "puppet_server" do |puppet|
  puppet.puppet_server = 'serverc.lab.example.com'
  puppet.options = '--onetime --verbose --waitforcert=10 --no-usecacheonfailure
 --no-daemonize'
end
```

   2.3. Launch the Vagrant machine and verify that it successfully registers with the Puppet master. Confirm that its applied configuration has enabled and started the **httpd** service.

```
[root@serverb webapp]# vagrant up
```

```
Bringing machine 'default' up with 'libvirt' provider...
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings..
... Output omitted ...
==> default: Running provisioner: puppet_server...
==> default: Running Puppet agent...
... Output omitted ...
==> default: Info: Caching catalog for dev.lab.example.com
==> default: Info: Applying configuration version '1444926117'
==> default: Notice: /Stage[main]/Main/Node[dev.lab.example.com]/Package[httpd]/
 ensure: created
==> default: Notice: /Stage[main]/Main/Node[dev.lab.example.com]/Service[httpd]/
 ensure: ensure changed 'stopped' to 'running'
==> default: Info: /Stage[main]/Main/Node[dev.lab.example.com]/Service[httpd]:
 Unscheduling refresh on Service[httpd]
==> default: Info: Creating state file /var/lib/puppet/state/state.yaml
==> default: Notice: Finished catalog run in 28.26 seconds
```

2.4. Verify that Puppet has installed the *httpd* package. Also verify that Puppet has enabled and started the **`httpd`** service.

```
[root@serverb webapp]# vagrant ssh
[vagrant@dev ~]$ rpm -q httpd
httpd-2.4.6-31.el7.x86_64
[vagrant@dev ~]$ systemctl is-active httpd
active
[vagrant@dev ~]$ systemctl is-enabled httpd
enabled
[vagrant@dev ~]$ exit
logout
Connection to 192.168.121.88 closed.
```

2.5. Use **`git diff`** to display the changes you made to the **`Vagrantfile`**. Check your changes into the local repository with a descriptive log message.

```
[root@serverb webapp]# git diff
diff --git a/webapp/Vagrantfile b/webapp/Vagrantfile
index cb66841..5b2ff35 100644
--- a/webapp/Vagrantfile
+++ b/webapp/Vagrantfile
@@ -10,4 +10,9 @@ Vagrant.configure(2) do |config|
     sudo yum install -y puppet
   SHELL

+  config.vm.provision "puppet_server" do |puppet|
+    puppet.puppet_server = 'serverc.lab.example.com'
+    puppet.options = '--onetime --verbose --waitforcert=10 --no-usecacheonfailu
+  end
+
 end
[root@serverb webapp]# git commit -a -m 'Configured puppet agent.'
... Output omitted ...
```

3. Archive the newly obtained Puppet certificates and make them available for reuse when the Vagrant machine is recreated.

3.1. Create a directory on the host to archive the Puppet certificates.

```
[root@serverb webapp]# mkdir -p puppet/ssl
```

3.2. Display the network and SSH information for the Vagrant machine to obtain its IP address.

```
[root@serverb webapp]# vagrant ssh-config
 Host default
  HostName 192.168.121.88
  User vagrant
  Port 22
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile /root/vagrant/webapp/.vagrant/machines/default/libvirt/
private_key
  IdentitiesOnly yes
  LogLevel FATAL
```

3.3. Copy the contents of the **/var/puppet/ssl** directory on the Vagrant machine to the archive directory on the host. Be sure to substitute the Vagrant machine's IP address in the command with the one displayed by the **vagrant ssh-config** command. The **--rsync-path** option to the **rsync** command will allow the file copy to be performed with sudo privilege on the Vagrant machine. The **-e** option to the **rsync** command specifies that the command will the performed through an SSH connection established with key-based authentication.

```
[root@serverb webapp]# rsync -avz --rsync-path='sudo rsync' \
-e 'ssh -i /root/vagrant/webapp/.vagrant/machines/default/libvirt/private_key' \
vagrant@192.168.121.88:/var/lib/puppet/ssl/ puppet/ssl/
The authenticity of host '192.168.121.88 (192.168.121.88)' can't be established.
ECDSA key fingerprint is 51:f7:72:0d:2d:e2:13:1b:84:19:61:41:6a:e5:d4:ac.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.121.88' (ECDSA) to the list of known hosts.
receiving incremental file list
./
crl.pem
certificate_requests/
certificate_requests/dev.lab.example.com.pem
certs/
certs/ca.pem
certs/dev.lab.example.com.pem
private/
private_keys/
private_keys/dev.lab.example.com.pem
public_keys/
public_keys/dev.lab.example.com.pem

sent 148 bytes  received 8723 bytes  2534.57 bytes/sec
total size is 10624  speedup is 1.20
```

3.4. Shut down the Vagrant machine.

```
[root@serverb webapp]# vagrant destroy
```

4. Modify the Vagrant configuration file so that the contents of the **/root/vagrant/webapp/puppet/ssl** directory are synced to the **/var/lib/puppet/ssl** of the Vagrant machine when it is launched. Launch the Vagrant machine and verify that **rsync** copies the folder and that the machine can apply the Puppet web server module.

   4.1. Add the following line within the **Vagrant.configure** code block in **Vagrantfile**. The line should be inserted immediately below the "# Define sync folder(s)" comment.

```
config.vm.synced_folder "puppet/ssl", "/var/lib/puppet/ssl", type: "rsync",
 owner: "puppet", group: "puppet", rsync__rsync_path: "sudo rsync"
```

   4.2. Verify the new configuration by launching a new Vagrant machine. The output should show **rsync** copying **/root/vagrant/webapp/puppet/ssl/** to **/var/lib/puppet/ssl** and the Puppet module should be applied.

```
[root@serverb webapp]# vagrant up
Bringing machine 'default' up with 'libvirt' provider...
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings...
... Output omitted ...
==> default: Rsyncing folder: /root/vagrant/webapp/puppet/ssl/ => /var/lib/
puppet/ssl
... Output omitted ...
==> default: Running provisioner: puppet_server...
==> default: Running Puppet agent...
==> default: Info: Retrieving plugin
==> default: Info: Caching catalog for dev.lab.example.com
==> default: Info: Applying configuration version '1444926117'
==> default: Notice: /Stage[main]/Main/Node[dev.lab.example.com]/Package[httpd]/
ensure: created
==> default: Notice: /Stage[main]/Main/Node[dev.lab.example.com]/Service[httpd]/
ensure: ensure changed 'stopped' to 'running'
==> default: Info: /Stage[main]/Main/Node[dev.lab.example.com]/Service[httpd]:
 Unscheduling refresh on Service[httpd]
==> default: Info: Creating state file /var/lib/puppet/state/state.yaml
==> default: Notice: Finished catalog run in 28.20 seconds
```

   4.3. Check your changes into the local Git repository with a descriptive log message.

```
[root@serverb webapp]# git add Vagrantfile puppet
[root@serverb webapp]# git commit -m 'Preserved Puppet SSL keys.'
... Output omitted ...
```

5. Configure the Vagrant machine to host a web application and verify the application.

   5.1. Create a directory on the host for the web application source code. Populate it with a simple HTML file, **index.html**, with the content, "**Welcome to Web App 1.0**".

```
[root@serverb webapp]# mkdir www
[root@serverb webapp]# echo 'Welcome to Web App 1.0' > www/index.html
```

   5.2. Modify the Vagrant configuration file so the source for the web application is synced to **/var/www/html** on the Vagrant machine upon its initialization. Set the type to **rsync**, set the ownership to user and group **apache**, and make sure it uses **sudo**

with **rsync**. The line should be inserted in the "# Define sync folder(s)" section of the **Vagrantfile**.

```
# Define sync folder(s)
config.vm.synced_folder "puppet/ssl", "/var/lib/puppet/ssl", type: "rsync",
 owner: "puppet", group: "puppet", rsync__rsync_path: "sudo rsync"
config.vm.synced_folder "www", "/var/www/html", type: "rsync", owner: "apache",
 group: "apache", rsync__rsync_path: "sudo rsync"
```

5.3. The files in the **/var/www/html** directory may not have the correct SELinux context after they have been copied from the host. Extend the existing Vagrant configuration file shell provisioner to execute **restorecon** to correct this.

```
# Define shell provisioner
config.vm.provision "shell", inline: <<-SHELL
  sudo cp /home/vagrant/sync/etc/yum.repos.d/* /etc/yum.repos.d
  sudo yum install -y puppet
  sudo restorecon -rv /var/www/html
SHELL
```

5.4. Modify the Vagrant configuration file host settings so port **80** of the Vagrant machine can be accessed via port **8000** on **localhost**. Add the following line within the **Vagrant.configure** code block.

```
# Define host settings
config.vm.hostname = "dev.lab.example.com"
config.vm.network :forwarded_port, guest: 80, host: 8000
```

5.5. Reload the Vagrant configuration for the newly introduced changes to take effect.

```
[root@serverb webapp]# vagrant reload
==> default: Halting domain...
==> default: Starting domain.
==> default: Waiting for domain to get an IP address...
==> default: Waiting for SSH to become available...
==> default: Creating shared folders metadata...
==> default: Forwarding ports...
==> default: 80 => 8000 (adapter eth0)
==> default: Rsyncing folder: /root/vagrant/webapp/ => /home/vagrant/sync
==> default: Rsyncing folder: /root/vagrant/webapp/puppet/ssl/ =>
 /var/lib/puppet/ssl
==> default: Rsyncing folder: /root/vagrant/webapp/www/ => /var/www/html
==> default: Machine already provisioned. Run `vagrant provision` or use
 the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always
 will still run.
```

5.6. Verify that the web application is working properly on the Vagrant machine using port 8000 on **localhost**.

```
[root@serverb webapp]# curl http://localhost:8000
Welcome to Web App 1.0
```

5.7. Check your changes into the local Git repository with a descriptive log message.

```
[root@serverb webapp]# git add Vagrantfile www
[root@serverb webapp]# git commit -m 'Synced web content from host.'
... Output omitted ...
```

6. Add a push component to the Vagrant configuration. It should deploy the web application's source to **/var/www/html** on the production web server running on **servera**. Also run **restorecon** to restore the SELinux contexts. Define it immediately after the Puppet provisioner defined in the **Vagrantfile**.

```
config.push.define "local-exec" do |push|
  push.inline = <<-SCRIPT
    rsync -avz www/ servera:/var/www/html/
    ssh servera 'restorecon -rv /var/www/html'
  SCRIPT
end
```

7. Check your changes into the local Git repository with a descriptive log message, then push all of your changes to the central Git repository.

```
[root@serverb webapp]# git commit -a -m 'Added a push component.'
[master 2174692] Added a push component.
 1 file changed, 7 insertions(+)
[root@serverb webapp]# git push
Counting objects: 33, done.
Compressing objects: 100% (23/23), done.
Writing objects: 100% (30/30), 10.01 KiB | 0 bytes/s, done.
Total 30 (delta 6), reused 0 (delta 0)
To ssh://student@workstation/var/git/vagrant.git
   a413d3c..2174692  master -> master
```

Lab: Implementing Puppet in a DevOps Environment

In this lab, you will use Puppet and Vagrant to create a development environment.

Resources	
Files	**/root/vagrant**
Application URL	**serverc:/var/git/webapp.git**
Machines	**servera**, **serverb**, **serverc**

Outcome(s)
You should be able to create a Puppet-configured Vagrant environment for development and deployment of a web application to a production server.

Before you begin
Reset your **servera**, **serverb** and **serverc** machines.

1. Log into **workstation** as **student** and run the lab setup script. The script will prepare **servera** for website hosting and make the Vagrant software available on **serverb**. It will also set up a Puppet master and Git repository on **serverc**.

```
[student@workstation ~]$ lab puppet-vagrant-lab setup
```

2. Log into the development server, **serverb**, as **root**. Install the *git* package. Configure Git for the **root** user, using "Student User" as the user name and "root@serverb.lab.example.com" as the email address. Set the default **push** behavior for Git to "simple".

3. Create a Vagrant working directory at **/root/vagrant** and then use the sync folder feature configured in **Vagrantfile** to clone into it the web application Git project located at **/var/git/webapp.git** on **serverc**.

4. Create the web application development environment by launching the Vagrant machine configured in the **webapp** Git project. This will configure TCP/8000 on **serverb** to forward to TCP/80 on the Vagrant machine. Verify the web application is functional.

5. Modify the web application source file, **/root/vagrant/webapp/www/index.html**, so it contains the content "**Welcome to Web App 2.0**". Sync the modified file to the Vagrant machine, verify the change, then commit the changes to the Git repository on **serverc**.

6. Prepare the production server, **servera**, to use the the standard configuration published by Puppet. Register it to the Puppet master on **serverc** and then apply its configuration.

7. Deploy the web application source code from the development server, **serverb**, to the production server, **servera**.

8. Run the lab grading script from **workstation** to check your work.

```
[student@workstation ~]$ lab puppet-vagrant-lab grade
```

Solution

In this lab, you will use Puppet and Vagrant to create a development environment.

Resources	
Files	**/root/vagrant**
Application URL	**serverc:/var/git/webapp.git**
Machines	**servera**, **serverb**, **serverc**

Outcome(s)
You should be able to create a Puppet-configured Vagrant environment for development and deployment of a web application to a production server.

Before you begin
Reset your **servera**, **serverb** and **serverc** machines.

1. Log into **workstation** as **student** and run the lab setup script. The script will prepare **servera** for website hosting and make the Vagrant software available on **serverb**. It will also set up a Puppet master and Git repository on **serverc**.

```
[student@workstation ~]$ lab puppet-vagrant-lab setup
```

2. Log into the development server, **serverb**, as **root**. Install the *git* package. Configure Git for the **root** user, using "Student User" as the user name and "root@serverb.lab.example.com" as the email address. Set the default **push** behavior for Git to "simple".

   2.1. Install the *git* package on **serverb**.

```
[root@serverb ~]# yum -y install git
```

   2.2. Configure the Git user identity for **root**.

```
[root@serverb ~]# git config --global user.name 'Student User'
[root@serverb ~]# git config --global user.email 'root@serverb.lab.example.com'
```

   2.3. Set the default **push** behavior for Git.

```
[root@serverb ~]# git config --global push.default simple
```

3. Create a Vagrant working directory at **/root/vagrant** and then use the sync folder feature configured in **Vagrantfile** to clone into it the web application Git project located at **/var/git/webapp.git** on **serverc**.

   3.1. Create the Vagrant working directory **/root/vagrant/**.

```
[root@serverb ~]# mkdir /root/vagrant
```

   3.2. Clone the **webapp.git** project into the Vagrant working directory.

```
[root@serverb ~]# cd /root/vagrant
[root@serverb vagrant]# git clone serverc:/var/git/webapp.git
Cloning into 'webapp'...
The authenticity of host 'serverc (172.25.250.12)' can't be established.
ECDSA key fingerprint is 85:57:bb:68:73:68:35:76:57:eb:bb:2b:7b:2c:2a:ad.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'serverc,172.25.250.12) to the list of known hosts.
root@serverc's password: redhat
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 22 (delta 2), reused 22 (delta 2)
Receiving objects: 100% (22/22), 9.20 KiB | 0 bytes/s, done.
Resolving deltas: 100% (2/2), done.
```

4. Create the web application development environment by launching the Vagrant machine configured in the **webapp** Git project. This will configure TCP/8000 on **serverb** to forward to TCP/80 on the Vagrant machine. Verify the web application is functional.

   4.1. Change to the **webapp** Git project directory.

   ```
   [root@serverb vagrant]# cd webapp
   ```

   4.2. Launch the Vagrant machine.

   ```
   [root@serverb webapp]# vagrant up
   ```

   4.3. Verify that the development web application is functional.

   ```
   [root@serverb webapp]# curl http://localhost:8000
   Welcome to Web App 1.0
   ```

5. Modify the web application source file, **/root/vagrant/webapp/www/index.html**, so it contains the content "**Welcome to Web App 2.0**". Sync the modified file to the Vagrant machine, verify the change, then commit the changes to the Git repository on **serverc**.

   5.1. Modify the **index.html** file.

   ```
   [root@serverb webapp]# echo 'Welcome to Web App 2.0' > www/index.html
   ```

   5.2. Sync the modified file to the Vagrant machine.

   ```
   [root@serverb webapp]# vagrant rsync
   ==> default: Rsyncing folder: /root/vagrant/webapp/ => /home/vagrant/sync
   ==> default: Rsyncing folder: /root/vagrant/webapp/puppet/ssl/ => /var/lib/
   puppet/ssl
   ==> default: Rsyncing folder: /root/vagrant/webapp/www/ => /var/www/html
   ```

   5.3. Verify the change to the development web application.

   ```
   [root@serverb webapp]# curl http://localhost:8000
   ```

```
Welcome to Web App 2.0
```

5.4. Commit the changes to the remote Git repository.

```
[root@serverb webapp]# git add www/index.html
[root@serverb webapp]# git commit -m 'Update web app to version 2.0'
[master 365254c] Update web app to version 2.0
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@serverb webapp]# git push
root@serverc's password: redhat
Counting objects: 7, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 347 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To serverc:/var/git/webapp.git
   58e97d5..365254c  master -> master
```

6. Prepare the production server, **servera**, to use the the standard configuration published by Puppet. Register it to the Puppet master on **serverc** and then apply its configuration.

6.1. Install the *puppet* package on **servera**.

```
[root@servera ~]# yum -y install puppet
```

6.2. Configure **serverc** as the Puppet master.

```
[root@servera ~]# echo 'server = serverc.lab.example.com' >> /etc/puppet/
puppet.conf
```

6.3. Initialize the Puppet client with a request to the Puppet master to sign the certificate request.

```
[root@servera ~]# systemctl start puppet.service
[root@servera ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service' '/etc/systemd/system/multi-
user.target.wants/puppet.service'
```

6.4. Apply the Puppet configuration for **servera**.

```
[root@servera ~]# puppet agent --test
```

7. Deploy the web application source code from the development server, **serverb**, to the production server, **servera**.

7.1. From **serverb**, connect to the web server on **servera** to view the website prior to the deployment of source from the development environment.

```
[root@serverb webapp]# curl http://servera
... Output omitted ...
```

Initially the Apache test page displays because the application has not been deployed.

7.2. Deploy the web application source code from **serverb** to **servera** using the **push** feature configured in **Vagrantfile**.

```
[root@serverb webapp]# vagrant push
The authenticity of host 'servera (172.25.250.10)' can't be established.
ECDSA key fingerprint is 85:57:bb:68:73:68:35:76:57:eb:bb:2b:7b:2c:2a:ad.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'servera,172.25.250.10' (ECDSA) to the list of known
 hosts.
root@servera's password: redhat
sending incremental file list
./
index.html

sent 111 bytes  received 34 bytes  19.33 bytes/sec
total size is 23  speedup is 0.16
root@servera's password: redhat
```

7.3. Verify that the website on **servera** now runs on the new source code. The output should be the same as the output previously retrieved when tested locally on **serverb**.

```
[root@serverb webapp]# curl http://servera
Welcome to Web App 2.0
```

8. Run the lab grading script from **workstation** to check your work.

```
[student@workstation ~]$ lab puppet-vagrant-lab grade
```

Summary

In this chapter, you learned:

- The Vagrant software requires a box, a provider, and a **`Vagrantfile`** to create a Vagrant machine.
- **`Vagrantfile`** is used to configure a Vagrant machine.
- Vagrant provisioners automate software installation and system configuration after a Vagrant machine has been built from a box image.
- Vagrant's **`puppet_server`** provisioner automates the integration of a Vagrant machine in an organization's Puppet infrastructure.
- Code deployment routines can be defined in **`Vagrantfile`** and invoked with the **`vagrant push`** command.

CHAPTER 14

IMPLEMENTING PUPPET IN RED HAT SATELLITE 6

Overview	
Goal	Implement Puppet in a Red Hat Satellite 6 environment.
Objectives	• Create a Puppet repository and upload a Puppet module. • Connect Puppet clients to the Satellite server. • Standardize configurations with Host Groups. • Implement smart class parameters to configure systems.
Sections	• Creating a Puppet Repository (and Guided Exercise) • Connecting Puppet Clients (and Guided Exercise) • Standardizing Configurations with Host Groups (and Guided Exercise) • Implementing Smart Class Parameters (and Guided Exercise)
Lab	• Implementing Puppet in Red Hat Satellite 6

Creating a Puppet Repository

Objectives

After completing this section, students should be able to:

- Create a custom product in Red Hat Satellite.
- Create a Puppet repository in Red Hat Satellite.
- Upload Puppet modules into a Puppet repository.

Custom products and repositories

Red Hat Satellite 6 uses Puppet to configure the hosts that it manages. Each Satellite or Capsule server acts as a Puppet master for client systems. Satellite publishes Puppet classes to the client hosts. Although classes are the basic unit of system configuration, they are loaded into the Satellite server as Puppet modules.

Note

Red Hat Satellite Capsule servers extend a Red Hat Satellite server in large scale environments. They provide software content and serve as Puppet masters for the Satellite clients they manage.

When Red Hat software content is added to a Satellite server, Red Hat repositories, and their parent products, are automatically created. However, to host custom software content or Puppet modules on a Satellite server, administrators must create repositories and products. Products are simply collections of repositories grouped by the administrator's preference. This allows an administrator to group software repositories from different software vendors, or related Puppet modules, into separate products.

As is the case with Red Hat content, custom products and repositories are content entities on Satellite Server. They are created and maintained in an organizational context. Products and repositories created under an organizational context will be visible only to that organization.

Creating custom products

Custom software and Puppet modules ackages are hosted in repositories on Satellite Server. Repositories cannot exist as their own entities and must be created within a product; therefore, administrators must begin with product creation.

Administrators can create products with the following steps executed as the *admin* user in the Satellite web UI. The following example demonstrates the creation of a product in the *ACME_Corporation* organization.

1. Click **Any Context** > **Any Organization** > **Acme_Corporation** from the context menu in the top-left corner to enter the *Acme_Corporation* organization context.
2. Click **Content** > **Products**.
3. Click the **New Product** button.
4. Enter the following information for the product.

4.1. Enter a name in the required **Name** field.

4.2. Enter a label in the required **Label** field.

4.3. The **GPG Key** field is used when verifying software packages. It is not relevant when creating a product to manage Puppet modules.

4.4. The **Sync Plan** field is used when maintaining software packages. It is not relevant when creating a product to manage Puppet modules.

4.5. Optionally enter a more thorough product description in the **Description** field.

5. Click the **Save** button.

Creating a Puppet repository

Most Satellite users are familiar with how Satellite publishes software to client systems. Yum repositories contain software packages that client systems can install. Similarly, Red Hat Satellite 6 introduced the concept of Puppet repositories, which are collections of Puppet modules. Managing Puppet repositories in Red Hat Satellite is similar to how Yum repositories are managed.

A Puppet repository has to be a part of a product within an organization in the Red Hat Satellite server. The repository can be created as part of an existing product, or a new product can be created solely for the purpose of having the Puppet repository.

The following steps describe how to create a Puppet repository in Red Hat Satellite:

1. Select the default organization. Click the **Any Context** tab at the left and select the appropriate organization from the pulldown menu.
2. Select the **Content** > **Products** tab to navigate to the products management screen.
3. Select or create a product to contain the Puppet repository.
4. Create the Puppet repository. Select the **Repositories** tab in the chosen product and click the **Create Repository** button.
5. Complete the form that appears. Fill in at least the name of the repository and select **puppet** for the repository type.
6. Click the **Save** button to confirm the selections and create the repository.

Once the Puppet repository is created, Puppet modules can be loaded into the Satellite server.

Uploading a Puppet module into a Puppet repository

The following steps describe how to import a Puppet module into a Puppet repository in Red Hat Satellite:

1. Select the organization that contains the Puppet repository that should contain the module to be uploaded.
2. Select the **Content** > **Products** tab.
3. Select the link corresponding to the product that contains the Puppet repository.
4. Select the link corresponding to the repository.

5. In the **Upload Puppet Module** box, click the **Browse** button.

6. Navigate and select the file that contains the Puppet module, then click the **Upload** button.

References

For more information, see

Red Hat Satellite 6.1 User Guide
https://access.redhat.com/documentation/en-US/Red_Hat_Satellite/6.1/html/User_Guide/index.html

Guided Exercise: Creating a Puppet Repository

In this practice exercise, you will create a Puppet repository that will contain the modules necessary for managing the NTP service on a Red Hat Enterprise Linux 7 host.

Resources	
Application URL	**http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz** and **http://materials.example.com/modules/puppetlabs-stdlib-4.9.0.tar.gz**
Machines	**satellite**

Outcome(s)
You should be able to create a new product that provides access to a Puppet repository in Red Hat Satellite 6. You should also be able to upload Puppet modules into a Red Hat Satellite 6 Puppet repository.

Before you begin
You should have a running Red Hat Satellite 6 server.

1. Log into **workstation** as **student** and run the lab setup script.

   ```
   [student@workstation ~]$ lab puppet-satellite setup
   ```

   This creates the **Operations** organization and other Red Hat Satellite entities that are needed to do the practice exercises for this chapter.

2. Download the **chrony** Puppet module to a local directory. Launch a web browser and navigate to **http://materials.example.com/modules**. Right-click the name of the **rht-chrony** Puppet module and save it to a local file.

3. The **rht-chrony** module has a dependency. It uses functions provided by the **puppetlabs-stdlib** Puppet module.

   Download the **stdlib** Puppet module to a local directory. Launch a web browser and navigate to **http://materials.example.com/modules**. Right-click the name of the **puppetlabs-stdlib** module and save it to a local file.

4. Open a web browser and navigate to **http://satellite.lab.example.com**. Log into the Satellite web UI as **admin** using the password **redhat**.

5. Select **Operations** as the default organization by clicking the **Any Context** tab at the left. Select **Operations** from the pulldown menu that appears.

6. Select the **Content** > **Products** tab to navigate to the *Products* management screen.

7. Create a product to contain the Puppet repository.

   7.1. Click the **New Product** button.

   7.2. Complete the form that appears. Fill in the fields according to the following table:

Field	Value
Name	Operations Product
Label	Operations_Product (autogenerated)

Leave the remaining fields unchanged or blank.

7.3. Click the **Save** button to confirm your selections and create the product.

8. Create the Puppet repository.

8.1. Select the **Repositories** tab, then click the **Create Repository** button.

8.2. Complete the form that appears. Fill in the fields according to the following table:

Field	Value
Name	Operations Modules
Label	Operations_Modules (autogenerated)
Type	puppet

Leave the remaining fields unchanged or blank.

8.3. Click the **Save** button to confirm your selections and create the repository.

9. Upload the **`rht-chrony`** Puppet module into the repository.

9.1. Select the **Operations Modules** link.

9.2. In the **Upload Puppet Module** box, click the **Browse...** button.

9.3. Navigate and select the file that contains the **`chrony`** Puppet module, **`rht-chrony-0.1.0.tar.gz`**.

9.4. Click the **Upload** button to upload the Puppet module.

10. Perform the previous steps to upload the **`puppetlabs-stdlib`** Puppet module into the repository.

11. There are a couple of ways to verify that the uploads succeeded. As each upload finished, a dialog box appeared in the **Upload Puppet Module** box saying "Content successfully uploaded".

Another way to confirm the uploads succeeded is to navigate to the *Products* management screen and select the **Operations Product** link. With the **Repositories** tab selected, the **Content** column should display a link, **2 Puppet Modules**. Clicking that link will display information about the **`chrony-0.1.0`** and **`stdlib-4.9.0`** modules.

Connecting Puppet Clients

Objectives

After completing this section, students should be able to:

- Create a content view that supports Puppet configuration.
- Create an activation key that enables Puppet configuration.
- Register a Puppet agent to a Red Hat Satellite server.

Installing and configuring Puppet agent

After Puppet modules have been uploaded into a Red Hat Satellite server (or capsule), the Puppet agent software needs to be deployed and configured on client systems to work with the Satellite server, which acts as the Puppet master. The following general steps must be taken to accomplish Puppet agent configuration:

1. Define a content view that provides necessary packages and Puppet modules.
2. Register the client with Red Hat Satellite.
3. Install Puppet agent software.
4. Sign the client host certificate.
5. Launch the Puppet agent.

Define a content view that provides necessary packages and Puppet modules

Red Hat Satellite content views determine which software packages and which Puppet modules are available for installation on client systems. Since Puppet may install software packages, the content view associated with a system should provide the necessary packages, including dependencies, that Puppet will use to configure services on a system. A content view does not determine how a system will be configured; instead, it provides access to the packages needed by Puppet to configure a system.

Although defining content views that provides access to software packages is beyond the scope of this course, it is useful to know what software should be made available to Puppet client systems when using a Satellite server. A content view must be created that provides access to the *puppet* package and its dependencies. These packages are provided by the **`Red Hat Satellite Tools 6.1`** and the **`Red Hat Enterprise Linux`** repositories at a minimum.

Consideration should also be given to additional software required by Puppet classes. Puppet classes can include resources that install software. For example, a Puppet class that configures a web server could install the *httpd* package and its dependencies.

Important

Content filters can be created in Red Hat Satellite that limit which packages client hosts have access to. Take care when defining content filters. Remember that Puppet classes must have access to the packages they will install to configure the client system to provide functionality.

Content views also determine which Puppet modules client hosts have access to. The Puppet classes in a module are not activated when they are defined in a content view, they are just made available. When editing a content view, select the **Puppet Modules** tab. The **Add New Module** button allows modules that have been previously loaded into the Satellite server to be chosen for publication to the client hosts. After the content view is created, it must be published and promoted to the software environment that the client will be registered with.

After a content view has been created, published, and promoted, Red Hat Satellite 6 creates a Puppet environment for that content view in each of the lifecycle environments in the organization. Puppet environment names have the following structure:

```
KT_ORG_ENV_VIEW_#
```

where ***ORG*** is the organization name, ***ENV*** is the lifecycle environment, ***VIEW*** is the content view name, and ***#*** is an internal sequence number. Some menu items will not be active when assigning a host group to a host if it has not been assigned a Puppet environment.

Register the client with Red Hat Satellite

After a content view has been created for the Puppet client, the client should be registered to the Red Hat Satellite server so that it is associated with that content view. The best way to automate this process is with an activation key.

When creating an activation key for the Puppet clients, the key should tie the clients to the software environment and the content view, that provides the desired Puppet modules. After selecting the appropriate organization context, select **Content** > **Activation keys**. Select the **Details** tab of an existing activation key or click the **New Activation Key** button. This will display the form where the software environment, then the content view can be selected.

Before registering the client system, the RPM that provides the CA certificate used to sign the Satellite server host certificate must be installed on the client. Once that is in place, the **`subscription-manager`** command registers the system to an organization with the activation key just created.

```
[root@host ~]# yum -y install http://SATELLITE.FQDN/pub/katello-ca-consumer-
latest.noarch.rpm
[root@host ~]# subscription-manager register --org=ORG --activationkey='KEY'
```

Install Puppet agent software

Use **`yum`** to install the Puppet agent. Another package that is usually installed at registration time is *katello-agent*. The Katello Agent is the software that allows Red Hat Satellite to perform remote software management of the client host.

```
[root@host ~]# yum -y install puppet katello-agent
```

This should be done after the client system is registered with an activation key and can access the appropriate Yum repositories. The packages can also be installed ahead of time when building virtual machine base images.

Note

The Red Hat Satellite server in our classroom environment has not been configured to provide software packages. Yum has been configured to access repositories that are not a part of the Satellite installation.

Once the Puppet agent software is installed, the primary Puppet configuration file must be modified to configure the Puppet agent to use the Satellite server/capsule as a Puppet master. Add the following line to **/etc/puppet/puppet.conf**:

```
server=satellite.FQDN
```

Sign the client host certificate
Finishing the client registration transaction takes two additional steps: Connect with Puppet master, then sign the Puppet agent host certificate. Have the Puppet agent contact the Puppet master. This can be done manually with the **puppet** command:

```
[root@host ~]# puppet agent --test --noop
```

The Puppet agent will send a host certificate to the Puppet master to allow for secure communications. By default the Puppet agent will connect to the master every 120 seconds and ask it to sign its host certificate request until it gets a signed certificate back. The default delay can be changed with the **--waitforcert=*DELAY*** option to **puppet agent**.

Log into the Satellite interface and take the following steps to sign the host certificate presented by the Puppet agent.

1. Managing Puppet certificates is not a function of a particular Red Hat Satellite organization. Set the organization context to **Any Organization**.
2. Select the **Infrastructure** > **Capsules** pulldown menu item.
3. Click the **Certificates** button at the right of the capsule server host name.
4. When the list of host certificates appears, click the **Sign** button at the right of the host that needs its certificate signed.

Red Hat Satellite can be configured to autosign Puppet host certificates. Once the list of host certificates for a capsule are displayed, click the **Autosign Entries** button that appears above the list. The screen that appears allows an administrator to create host name entries (that can include wildcards) that will have their host certificates signed automatically when they first connect to the Satellite server.

Warning

Configuring Red Hat Satellite to autosign Puppet host certificates creates a security risk. In this case, any host can connect and request Puppet manifests (which may contain privileged information, such as passwords, shared keys, and certificates).

Launch the Puppet agent

The following commands would start and enable the Puppet agent as a daemon on a Red Hat Enterprise Linux 7 system:

```
[root@host ~]# systemctl start puppet.service
[root@host ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service' '/etc/systemd/multi-user.target.wants/
puppet.service'
```

Guided Exercise: Connecting Puppet Clients

In this lab, you will create a content view and activation key to register Puppet clients to Red Hat Satellite.

Resources	
Application URL	**`http://satellite.lab.example.com/pub/katello-ca-consumer-latest.noarch.rpm`**
Machines	**`satellite`** and **`servera`**

Outcome(s)
You should be able to create a content view and activation key and use them to register client hosts to your Red Hat Satellite server so that they can deploy and use Puppet for automated system configuration.

Before you begin
Reset your **`servera`** server. You should have a running Red Hat Satellite 6 server with an existing organization called **`Operations`**. The organization should have a software lifecycle path that includes a lifecycle environment called **`Dev`**. A product called **`Operations Product`** should exist that includes a Puppet repository that provides the **`chrony`** and **`stdlib`** Puppet modules.

1. Create a content view for the client hosts. It should include the Puppet repository with the **`chrony`** and **`stdlib`** Puppet modules.

   1.1. Select **Any Context** > **Any Organization** > **Operations** to set the organization context to **`Operations`**.

   1.2. Select **Content** > **Content Views**, then click the **Create New View** button.

   Complete the form that appears with the following values:

Field	Value
Name	Operations Content
Label	Operations_Content (autogenerated)
Description	Content view for Operations servers
Composite View?	Unchecked

   Click the **Save** button to confirm.

   1.3. Click the **Puppet Modules** tab, then click the **Add New Module** button.

   Click the **Select a Version** buttons that correspond to the **`chrony`** and **`stdlib`** modules in the **Actions** column.

   Click the **Select Version** button for the **`Use Latest (currently 0.1.0)`** version of the **`chrony`** module. Do the same for the **`Use Latest (currently 4.9.0)`** version of the **`stdlib`** module.

   1.4. Publish the content view and promote it to the **`Dev`** environment.

Click the **Publish New Version** button, then click **Save** to confirm that you want to publish it.

Wait for the publish to complete, then click the **Promote** button in the **Actions** column for version 1.0 of the content view. Select the **Dev** environment, then click the **Promote Version** button.

When it completes, you should see both **Library** and **Dev** in the **Environments** column.

2. Confirm the proper locations are assigned to the Puppet environments for the **Operations** organization.

   2.1. If necessary, select **Operations** as the default organization context.

   2.2. Navigate to the **Puppet Environments** screen by selecting **Configure** > **Environments**.

   2.3. Select the environment for the **Operations** content library, **KT_Operations_Library_Operations_Content_#**.

   2.4. Click the **Locations** subtab and put **Boston** and **Default Location** in the **Selected items** window.

   2.5. Click **Submit** to apply your changes.

   2.6. Repeat the previous steps for the **Dev** Puppet environment of the **Operations** organization. Select the environment for the **Operations** development content, **KT_Operations_Dev_Operations_Content_#**.

   2.7. Click the **Locations** subtab and put **Boston** and **Default Location** in the **Selected items** window.

   2.8. Click **Submit** to apply your changes.

3. Create an activation key to register your Puppet client machines.

   3.1. Make sure **Operations** is selected as the current organization context.

   3.2. Select **Content** > **Activation keys**, then click the **New Activation Key** button.

   Complete the form that appears with the following values:

Field	Value
Name	Operations Host
Content Host Limit: Unlimited Content Hosts	Leave checked
Description	Activation key that registers Operations hosts
Environment	Select **Dev**
Content View	Select **Operations Content** from the pulldown menu

   Click **Save** to confirm.

3.3. Select the **Subscriptions** tab, uncheck the **Auto-Attach** checkbox, then click the **Save** button. This prevents Red Hat Satellite from automatically attaching software subscriptions to the content host that is registering with this activation key.

Click the **Add** subtab, then select the **Operations Product** subscription. Click the **Add Selected** button to confirm. This will give the client system access to the Puppet repositories provided by the product.

4. Configure the Puppet client host for Red Hat Satellite use. Log in as **root** on **servera** and register it with the **Operations Host** activation key.

```
[root@servera ~]# yum -y install http://satellite.lab.example.com/pub/katello-ca-
consumer-latest.noarch.rpm
... Output omitted ...
[root@servera ~]# subscription-manager register --org 'Operations' \
                     --activationkey 'Operations Host'
The system has been registered with ID: 3badb391-9eaa-448c-91f7-bc4fae059510

Installed Product Current Status:
Product Name: Red Hat Enterprise Linux Server
Status:       Not Subscribed

Unable to find available subscriptions for all your installed products.
```

subscription-manager was unable to find available subscriptions for the client. This is because the Red Hat Satellite server in our lab environment has not been configured with a manifest the provides Red Hat software subscriptions.

5. Use a browser to get into the Red Hat Satellite web interface and confirm the client host is registered in Red Hat Satellite.

5.1. Set the organization context to **Operations**.

5.2. Select **Hosts** > **Content Hosts**.

5.3. An entry for **servera.lab.example.com** should be in the list of hosts that appears. It should have a recent registration time.

6. Go back to the **root** session on the client system. Install Puppet and the Katello Agent software on the host.

```
[root@servera ~]# yum -y install puppet katello-agent
... Output omitted ...
Installed:
  katello-agent.noarch 0:2.2.5-1.el7sat      puppet.noarch 0:3.6.2-4.el7sat
Dependency Installed:
... Output omitted ...
Complete!
```

7. Configure Puppet to use the Red Hat Satellite server.

```
[root@servera ~]# echo 'server=satellite.lab.example.com' >> /etc/puppet/puppet.conf
```

8. Display what actions Puppet will take (do not apply them). Note that there is a certificate issue. We must solve that on the Satellite server.

```
[root@servera ~]# puppet agent --test --noop
Info: Creating a new SSL key for servera.lab.example.com
Info: Caching certificate for ca
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for servera.lab.example.com
Info: Certificate Request fingerprint (SHA256): E5:04:15:71:F6:9B:A7:EF:
 88:84:88:53:E1:C2:5F:9E:97:35:E3:22:31:5F:13:3A:10:8E:4F:0E:61:3E:AD:C5
Info: Caching certificate for ca
Exiting; no certificate found and waitforcert is disabled
```

9. Sign the host certificate for the client system in Satellite.

   9.1. Go back to the web browser with the Satellite user interface and select **Infrastructure** > **Capsules**.

   9.2. Click the **Certificates** button at the right of the **satellite.lab.example.com** capsule host name.

   9.3. A list of host certificates will appear. Click the **Sign** button at the right of the **servera** host name.

10. On the Satellite client system, display what actions Puppet will take (do not apply them). Note that there is a catalog issue, but that will be solved in the next practice exercise.

```
[root@servera ~]# puppet agent --test --noop
Info: Caching certificate for servera.lab.example.com
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for servera.lab.example.com
Warning: Unable to fetch my node definition, but the agent run will continue:
Warning: Error 400 on SERVER: Failed to find servera.lab.example.com via exec:
 Execution of '/etc/puppet/node.rb servera.lab.example.com' returned 1:
Info: Retrieving plugin
Error: /File[/var/lib/puppet/lib]: Could not evaluate: Could
 not retrieve information from environment production source(s)
 puppet://satellite.lab.example.com/plugins
Info: Caching catalog for servera.lab.example.com
Info: Applying configuration version '1441218498'
Notice: Finished catalog run in 0.07 seconds
```

Standardizing Configurations with Host Groups

Objectives

After completing this section, students should be able to:

- Define a standard configuration with host groups.
- View report and audit information about hosts configured by Puppet.

Standardizing systems with Puppet

Once Puppet modules are uploaded into a Puppet repository in Red Hat Satellite, the modules need to be associated with the client machines to be configured. This is accomplished through the use of host groups. A host group defines a set of default values that hosts inherit when placed in that group. Hosts can belong to only one host group. For example, there could be one host group that configures web servers in a datacenter. Another host group could configure DNS servers.

Once a host group is created, Puppet classes defined in the modules associated with the content view for that host group can be selected for deployment on client systems in that host group. These Puppet classes become active and configure the servers in the host group. They can create users, install software, deploy configuration files, start services, and many more tasks.

Puppet classes can depend upon functionality provided by other modules. Many Puppet classes use functionality provided by the **`stdlib`** class, provided by the **`puppetlabs-stdlib`** module. When defining host groups, the Satellite administrator must be aware of these dependencies and make sure they are satisfied when including classes in a host group. Configuration groups, which will be introduced later in this section, can simplify the deployment of related Puppet classes.

Creating a host group

Each Satellite organization has its own private collection of host groups. Set the organization context to the organization that the host group will belong to. Select **Configure** > **Host groups**, then click the **New Host Group** button. Specify the **Lifecycle Environment**, **Content View**, **Puppet Environment**, **Content Source**, **Puppet CA**, and **Puppet Master**. Choose a descriptive name for the host group, then click **Submit** to create the host group.

Once the host group has been created, the Puppet classes must be selected that define how the servers of that host group will be configured. Select the **Puppet Classes** tab. Expand the Puppet module links that contain the classes to include. Click the **+** to the right of the class names so they appear in the **Included Classes** column. When configuration groups have been defined, they can be selected as well. Selecting a configuration group will activate all of the Puppet classes in that configuration group.

Once the host group is created, it can be assigned to individual hosts. Set **`Any Organization`** as the default organization context, then select the **Hosts** > **All hosts** tab. Identify the host of interest in the list of hosts and click the host name link. This will bring up the Puppet details page for that server. Click the **Edit** button at the top of the screen, then select the desired host group from the **Host Group** pulldown menu.

A host group can be associated with machines that register with an activation key. When creating the host group, select the **Activation Key** tab and select the existing activation keys to associate with the host group.

Viewing Puppet audit and reporting activity

Once Puppet is configured to run as a daemon on the client, it checks in with the Red Hat Satellite server every half hour to see if there are changes to be made. All of that information is logged into the Satellite server's database so it can be audited.

To view information about the Puppet transactions for a host, change to the organization that host belongs to. Select **Hosts** > **All hosts**. Click the link for the host of interest to look at the Puppet details page for that host. This is a general dashboard of information about that host's Puppet activity.

Click the **Reports** button. Puppet actions that are applied to the host are numbered, and the numbers are links to a more detailed report about that action. When viewing a report, the **View Diff** link will display the differences between the before and after state of the Puppet transaction.

Configuration groups

Some Puppet modules contain Puppet classes that depend on other classes to perform their configurations. Configuration groups allow these classes to be grouped together so they can be managed as a unit, instead of individually.

Creating a configuration group

The following steps describe how to create a configuration group and add Puppet classes to it:

1. Select the organization that should include the configuration group.
2. Select the **Configure** > **Config groups** tab.
3. Click the **New Config Group** button.
4. Specify the name of the configuration group.
5. Click/expand the desired modules presented in the **Available Classes** column.
6. Click the **+** to the right of the classes to be included in the configuration group. As the classes are added, they will appear in the **Included Classes** column.
7. Click the **Submit** button to confirm the selections.

Guided Exercise: Standardizing Configurations with Host Groups

In this lab, you will create a host group that will define which Puppet modules get applied to all of the hosts in an organization.

Resources	
Files	**`/etc/chrony.conf`**
Machines	**`satellite`** and **`servera`**

Outcome(s)
You should be able to create a host group that defines a standard Puppet configuration that configures NTP on client hosts in an organization.

Before you begin
You should have a running Red Hat Satellite 6 server with an existing organization called **`Operations`**. The organization should have a software lifecycle path that include a lifecycle environment called **Dev**. A content view called **`Operations Content`** should exist that provides the **`chrony`** and **`stdlib`** Puppet modules.

1. Create a configuration group, called **`NTP Classes`**, in the **`Operations`** organization that includes the **`chrony`**, **`chrony::params`**, and **`stdlib`** Puppet classes.

   1.1. Select **Any Context** > **Any Organization** > **Operations** to set the organization context to **`Operations`**.

   1.2. Select **Configure** > **Config groups**.

   1.3. Click the **New Config Group** button. Enter **`NTP Classes`** in the **Name** field of the form that appears.

   1.4. Click the names of the **`chrony`** and **`stdlib`** modules to display their classes. Select the **`chrony`**, **`chrony::params`**, and **`stdlib`** classes so they display in the **Included Classes** column on the page.

   The **`rht-chrony`** Puppet module includes both the **`chrony`** and **`chrony::params`** classes. The **`chrony::params`** class defines the default values of the class parameters for the **`chrony`** class, so it is a dependency that must be included to support the use of **`chrony`**.

   1.5. Click the **Submit** button to confirm the selections and create the configuration group.

2. Create a host group called **`Operations Host Group`** in the **`Operations`** organization.

   2.1. Select **Configure** > **Host groups**, then click the **New Host Group** button.

   Complete the form that appears with the following values:

Field	Value
Name	Operations Host Group

Field	Value
Lifecycle Environment	Dev
Content View	Operations Content
Puppet Environment	KT_Operations_Dev_Operations_Content_#
Content Source	satellite.lab.example.com
Puppet CA	satellite.lab.example.com
Puppet Master	satellite.lab.example.com

2.2. Select the **Puppet Classes** tab.

Click the **Add** button next to the **`NTP Classes`** configuration group listed in the **Available Config Groups** column. This will select the configuration group and move it into the **Included Config Groups** column.

2.3. Click the **Network** tab, the **Operating System** tab, and the **Parameters** tab and explore their options, but do not make any changes. The options provided by the **Parameters** tab will be explored in another section.

2.4. Select the **Locations** tab. Move **`Boston`** and **`Default Location`** to the list of **Selected items**.

2.5. Select the **Organizations** tab. Make sure **`Operations`** is selected.

2.6. Click the **Submit** button at the bottom to confirm the selections and create the new host group.

3. Assign the new host group profile to **`servera.lab.example.com`**.

3.1. Set **`Any Organization`** as the default organization context.

3.2. Select the **Hosts** > **All hosts** tab.

3.3. Identify **`servera.lab.example.com`** in the list of hosts. Note that the **Host group** column is empty for this host. Click the checkbox selecting this host.

3.4. Click the **Select Action** pulldown menu, and choose the **Assign Organization** menu item. Click the **Select Organization** button, and choose **`Operations`** Select the **Fix Organization on Mismatch** button to override the existing organization assignment. Click **Submit** to commit the selections.

3.5. Click the **Select Action** pulldown menu, and choose the **Assign Location** menu item. Click the **Select Location** button, choose **`Boston`**, click the **Fix Location on Mismatch** button, then click **Submit** to commit the selections.

3.6. Click the **Select Action** pulldown menu, and choose the **Change Environment** menu item. Click the **Select environment** button, choose **`KT_Operations_Dev_Operations_Content_#`**, then click **Submit** to commit the selection.

3.7. Click the **Select Action** pulldown menu, and choose the **Change Group** menu item. Click the **Select host group** pulldown, choose **`Operations Host Group`**, then click **Submit** to commit the selection.

4. Confirm the client host sees the difference in its NTP configuration. Log into **servera** as **root** and note how Puppet changes the local environment and proposes a change to **/etc/chrony.conf**.

```
[root@servera ~]# puppet agent --test --noop
Warning: Local environment: "production" doesn't match server specified
 node environment "KT_Operations_Dev_Operations_Content_8", switching
 agent to "KT_Operations_Dev_Operations_Content_8".
... Output omitted ...
Info: Loading facts in /var/lib/puppet/lib/facter/facter_dot_d.rb
Info: Loading facts in /var/lib/puppet/lib/facter/root_home.rb
Info: Loading facts in /var/lib/puppet/lib/facter/puppet_vardir.rb
Info: Loading facts in /var/lib/puppet/lib/facter/pe_version.rb
Info: Caching catalog for servera.lab.example.com
... Output omitted ...
Info: Applying configuration version '1441288571'
Notice: /Stage[main]/Chrony/File[/etc/chrony.keys]/mode: current_value 0640,
 should be 0644 (noop)
Notice: /Stage[main]/Chrony/File[/var/log/chrony]/owner: current_value chrony,
 should be root (noop)
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/ensure: current_value link,
 should be file (noop)
Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Scheduling refresh of
 Service[chronyd]
Notice: /Stage[main]/Chrony/Service[chronyd]: Would have triggered 'refresh'
 from 1 events
Notice: Class[Chrony]: Would have triggered 'refresh' from 4 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.62 seconds
```

Note that the original configuration file is a symbolic link. This is part of our classroom's standard build.

```
[root@servera ~]# ls -l /etc/chrony.conf
lrwxrwxrwx. 1 root root 15 Aug 26 17:44 /etc/chrony.conf -> chrony.conf-rht
```

/etc/chrony.conf is the main **chrony** configuration file. Puppet will manage the file and recreate it.

5. Have Puppet apply the change to the host system.

```
[root@servera ~]# puppet agent --test
... Output omitted ...
Info: Applying configuration version '1441290722'
Notice: /Stage[main]/Chrony/File[/etc/chrony.keys]/mode: mode changed '0640'
 to '0644'
Notice: /Stage[main]/Chrony/File[/var/log/chrony]/owner: owner changed
 'chrony' to 'root'
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/ensure: defined content as
 '{md5}e814ce39347f1431253f933967bcdffb'
Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Scheduling refresh of
 Service[chronyd]
Notice: /Stage[main]/Chrony/Service[chronyd]: Triggered 'refresh' from 1
 events
Notice: Finished catalog run in 0.89 seconds
[root@servera ~]# ls -l /etc/chrony.conf
-rw-r--r--. 1 root root 928 Sep  3 10:32 /etc/chrony.conf
```

Puppet converted **/etc/chrony.conf** to a simple file that it will manage. Notice Puppet changed the ownership of the file from **chrony** to **root** and changed the permissions to 644.

6. Go back to the Red Hat Satellite web interface and audit the changes Puppet made to the configuration file.

   6.1. Set the organization context to **Operations**.

   6.2. Select **Hosts** > **All hosts**.

   6.3. Click the link for **servera.lab.example.com** to look at the Puppet details page for that host.

   6.4. Click the **Reports** button.

   6.5. Click the link for the report that has the **4** in the **Applied** column. A list of messages will show that the attributes and contents of **/etc/chrony.conf** have been changed. Look for the ownership and permission changes that were made to **/etc/chrony.conf**. The messages will also show that the **chrony** service was refreshed (restarted).

Implementing Smart Class Parameters

Objectives

After completing this section, students should be able to:

- Use smart class parameters to define, and override, Puppet module default values.

Introducing smart class parameters

Puppet parameterized classes allow variables to be defined with default values in the class definition header. When this type of class is used in a Puppet manifest or catalog, different values can be specified that override the default values.

Here is a simple example of a Puppet class declaration that takes parameters:

```
class ntp ($servers = undef, $enable = true, $ensure = 'running') {
  ...
```

$servers, **$enable**, and **$ensure** are parameters that can be passed into that class. If they are not specified in the catalog that uses this class, then **$servers** is undefined, **$enable** is set to true (probably activating the service), and **$ensure** is set to 'running' (causing the NTP service to be launched).

Here is a look at how the class could be used:

```
class { 'ntp':
    enable => false,
    ensure => 'stopped',
}
```

The title of the **class** determines which Puppet class is being used. The specified parameters override the default values originally specified in the Puppet class definition. In the previous example, the **enable** parameter is set to false and the **ensure** parameter is set to 'stopped'.

Using Puppet classes in Red Hat Satellite 6

In Red Hat Satellite 6, smart class parameters are a way to pass specific values into a Puppet module of a system that is being configured. When a module is imported, the default values are used initially and the class parameters are fixed. To override them, they have to be flagged as such. The values can be overridden based on hosts that match an expression based on facts about the host. The customized values can be additionally overridden on a host-by-host basis, via the Satellite 6 web interface.

To override the default values for smart class parameters, set the organization context to the organization with the relevant Puppet modules. Select **Configure** › **Puppet Classes**, then click the name displayed in the **Class name** column of the Puppet class to be manipulated.

The **Smart Class Parameter** subtab allows parameters to be tuned. When a parameter is selected, the **Override** checkbox enables the class parameter's default value to be changed. Changes can be made globally with a new **Default value**.

Matcher-Value expressions can be used to apply different values for smart class parameters to a limited number of hosts where one of the **Matcher** expressions is true. These expressions can

be based on Satellite/Foreman variables and values, or they can be based on system facts about the client host. **`Matcher`** expressions are comparisons of the form, ***`variable = value`***. Wild cards and regular expressions are not valid as **`Matcher`** expressions. Here are a couple of simple examples of **`Matcher`** expressions:

```
ipaddress = '192.168.10.250'

hostgroup = 'Web Servers'
```

Guided Exercise: Implementing Smart Class Parameters

In this exercise, you will use Puppet to deploy a standard NTP configuration to Red Hat Satellite client hosts.

Resources	
Files	**/etc/chrony.conf**
Machines	**satellite** and **servera**

Outcome(s)
You should be able to use Puppet smart class parameters in a Red Hat Satellite server to customize the NTP configuration of hosts in a specific organization.

Before you begin
You should have a running Red Hat Satellite 6 server with an existing organization called **Operations**. A client system, **servera.lab.example.com**, should be registered to the Satellite server and belong to a host group that provides configuration by the **chrony** Puppet module.

1. Adjust the class parameters for the **chrony** Puppet class so that the default NTP server is **0.rhel.pool.ntp.org**. The NTP server for hosts configured as **Operations Host Group** servers should be **classroom.example.com**.

   1.1. Open a browser to the Red Hat Satellite web user interface. Select **Any Context** > **Any Organization** > **Operations** to set the organization context to **Operations**.

   1.2. Select **Configure** > **Puppet classes**, then click **chrony** in the **Class name** column.

   1.3. Select the **Smart Class Parameter** subtab.

   1.4. Scroll, locate, and click the **servers** link at the left. That is the name of the class parameter. Check the **Override** box to allow for a custom value to be entered.

   1.5. The **chrony** class defines the **servers** parameter as an array of strings. The following text is found in the comments of the **init.pp** manifest in the Puppet module.

```
# [*servers*]
#   An array of strings containing the NTP servers chrony should use and the
#   additional configuration settings for each one.  Each entry in the array
#   starts with the server's ip address or fqdn followed by {any additional
#   options accepted by the chrony server directive}
#   [http://chrony.tuxfamily.org/manual.html#server-directive].
```

   Select **array** from the pulldown menu for the **Parameter type**. Set the **Default value** to the following:

```
["0.rhel.pool.ntp.org"]
```

1.6. Click the **Add Matcher-Value** button. Complete the form that appears with the following values:

Field	Value
Match	hostgroup=Operations Host Group
Value	["classroom.example.com"]

Click the **Submit** button to confirm the changes.

2. Log in as **root** on **servera** and dIsplay the original **server** configurations in **/etc/chrony.conf**. Confirm the client host sees the difference in its **chrony** configuration.

```
[root@servera ~]# grep '^server' /etc/chrony.conf
server 0.rhel.pool.ntp.org iburst
server 1.rhel.pool.ntp.org iburst
server 2.rhel.pool.ntp.org iburst
server 3.rhel.pool.ntp.org iburst
[root@servera ~]# puppet agent --test --noop
Warning: Local environment: "production" doesn't match server specified
 node environment "KT_Operations_Dev_Operations_Content_8", switching
 agent to "KT_Operations_Dev_Operations_Content_8".
Info: Retrieving plugin
... Output omitted ...
Info: Applying configuration version '1441292816'
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content:
--- /etc/chrony.conf    2015-09-03 10:32:04.471043566 -0400
+++ /tmp/puppet-file20150903-2429-91igrw        2015-09-03 11:06:57.671851980 -0400
@@ -1,7 +1,4 @@
-server 0.rhel.pool.ntp.org iburst
-server 1.rhel.pool.ntp.org iburst
-server 2.rhel.pool.ntp.org iburst
-server 3.rhel.pool.ntp.org iburst
+server classroom.example.com iburst


 # Ignore stratum in source selection.

Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content: current_value
 {md5}e814ce39347f1431253f933967bcdffb, should be
 {md5}345a7419db5320fd7c6c6c6395105070 (noop)
Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Scheduling refresh
 of Service[chronyd]
Notice: /Stage[main]/Chrony/Service[chronyd]: Would have triggered
 'refresh' from 1 events
Notice: Class[Chrony]: Would have triggered 'refresh' from 2 events
Notice: Stage[main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.68 seconds
```

Puppet intends to change the **chrony** server definition to **classroom.example.com**. It prefers this value instead of **0.rhel.pool.ntp.org** because the **Matcher-Value** pair matched the Puppet client based on its assigned host group.

3. Have Puppet apply the change to the host system, then redisplay the **server** configurations in **/etc/chrony.conf**.

```
[root@servera ~]# puppet agent --test
Warning: Local environment: "production" doesn't match server specified
```

```
 node environment "KT_Operations_Dev_Operations_Content_8", switching
 agent to "KT_Operations_Dev_Operations_Content_8".
Info: Retrieving plugin
Info: Loading facts in /var/lib/puppet/lib/facter/facter_dot_d.rb
Info: Loading facts in /var/lib/puppet/lib/facter/root_home.rb
Info: Loading facts in /var/lib/puppet/lib/facter/puppet_vardir.rb
Info: Loading facts in /var/lib/puppet/lib/facter/pe_version.rb
Info: Caching catalog for servera.lab.example.com
... Output omitted ...
Info: Applying configuration version '1441292816'
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content:
--- /etc/chrony.conf    2015-09-03 10:32:04.471043566 -0400
+++ /tmp/puppet-file20150903-2579-sl3kwe        2015-09-03 11:07:23.044912553 -0400
@@ -1,7 +1,4 @@
-server 0.rhel.pool.ntp.org iburst
-server 1.rhel.pool.ntp.org iburst
-server 2.rhel.pool.ntp.org iburst
-server 3.rhel.pool.ntp.org iburst
+server classroom.example.com iburst


 # Ignore stratum in source selection.

Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Filebucketed
 /etc/chrony.conf to puppet with sum e814ce39347f1431253f933967bcdffb
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content: content
 changed '{md5}e814ce39347f1431253f933967bcdffb' to
 '{md5}345a7419db5320fd7c6c6c6395105070'
Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Scheduling refresh
 of Service[chronyd]
Notice: /Stage[main]/Chrony/Service[chronyd]: Triggered 'refresh' from
 1 events
Notice: Finished catalog run in 0.97 seconds
[root@servera ~]# grep '^server' /etc/chrony.conf
server classroom.example.com iburst
```

Lab: Implementing Puppet in Red Hat Satellite 6

In this lab, you will create a Puppet repository for maintaining the Puppet modules that configure the NTP service on the Red Hat Enterprise Linux 7 servers in the **Finance** department. You will use Red Hat Satellite and Puppet to configure the NTP service for the registered **Finance** client systems.

Resources	
Application URL	**http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz** and **http://materials.example.com/modules/puppetlabs-stdlib-4.9.0.tar.gz**
Machines	**satellite** and **serverb**

Outcome(s)
You should be able to create a new product, with a Puppet repository, in Red Hat Satellite and populate it with existing Puppet modules. You should also be able to create a content view, an activation key, and a host group, and configure them so that Puppet configures the NTP service on client hosts.

Before you begin
Reset your **serverb** server. You should have a running Red Hat Satellite 6 Server with an existing organization called **Finance**. The organization should have a software lifecycle path that includes a lifecycle environment called **Dev**.

1. If you have not previously executed the lab setup script in the Creating a Puppet Repository practice exercise, do it now. Log into **workstation** as **student** and run the lab setup script.

   ```
   [student@workstation ~]$ lab puppet-satellite setup
   ```

   This creates the **Finance** organization and other Red Hat Satellite entities that are needed to do the lab.

2. Download the **chrony** and **stdlib** Puppet modules to a local directory.

3. Log into the Satellite web UI and make **Finance** the current organization.

4. Create a product called **Finance Product** to contain the Puppet repository.

5. Create the **Finance Modules** Puppet repository.

6. Upload the **chrony** and **stdlib** Puppet modules into the repository.

7. Create a content view called **Finance Content**. It must include the Puppet repository with the **chrony** and **stdlib** Puppet modules. Publish and promote the content view so it is available to Finance hosts that are assigned to the **Dev** software lifecycle environment.

8. Confirm the proper locations are assigned to the Puppet environments for the **Finance** organization.

9. Create an activation key in the **Finance** organization, called **Finance Host**, to register Puppet client machines. The **Finance Content** content view should be assigned to hosts that register with it and their software lifecycle environment should be **Dev**. Since the classroom Red Hat Satellite server does not provide software packages, the activation key should not auto attach to any Red Hat software subscriptions. The activation key should subscribe hosts to the custom **Finance Product** product so they have access to the Puppet repository it provides.

10. Configure the Puppet client on **serverb** for Red Hat Satellite use. Register it to the **Finance** organization with the **Finance Host** activation key.

11. Install Puppet and the Katello agent software on the Satellite client host, **serverb**.

12. Configure Puppet on **serverb** to use the Red Hat Satellite server as the Puppet master. Run **puppet** once in test mode so it sends a certificate to the Satellite server.

13. Use the Satellite web user interface to sign the host certificate for the client system.

14. Create a host group called **Finance Host Group** in the **Finance** organization.

15. Assign the **Finance Host Group** host group profile to **serverb.lab.example.com**. **serverb** should be assigned to the **Finance** organization and the **Boston** location. Apply the **Finance** content view to the host and make it a member of the **Dev** lifecycle environment. It should also be assigned to the **KT_Finance_Dev_Finance_Content_#** Puppet environment. The Satellite server should act as **serverb**'s content source, Puppet certificate authority, and Puppet master.

16. Define a smart class parameter that populates the **servers** parameter of the **chrony** Puppet class with a list of servers that only includes the **classroom.example.com** and **satellite.lab.example.com** hosts. This configuration should only apply to hosts that are members of the **Finance Host Group** host group.

17. Have Puppet apply the change to the host system to confirm the configuration is correct.

18. Persistently configure Puppet on **serverb** to run as a service.

19. Run the lab grading script from **workstation** to check your work. The grading program will access **serverb** to evaluate your work.

```
[student@workstation ~]$ lab puppet-satellite grade
```

Solution

In this lab, you will create a Puppet repository for maintaining the Puppet modules that configure the NTP service on the Red Hat Enterprise Linux 7 servers in the **Finance** department. You will use Red Hat Satellite and Puppet to configure the NTP service for the registered **Finance** client systems.

Resources	
Application URL	**http://materials.example.com/modules/rht-chrony-0.1.0.tar.gz** and **http://materials.example.com/modules/puppetlabs-stdlib-4.9.0.tar.gz**
Machines	**satellite** and **serverb**

Outcome(s)
You should be able to create a new product, with a Puppet repository, in Red Hat Satellite and populate it with existing Puppet modules. You should also be able to create a content view, an activation key, and a host group, and configure them so that Puppet configures the NTP service on client hosts.

Before you begin
Reset your **serverb** server. You should have a running Red Hat Satellite 6 Server with an existing organization called **Finance**. The organization should have a software lifecycle path that includes a lifecycle environment called **Dev**.

1. If you have not previously executed the lab setup script in the Creating a Puppet Repository practice exercise, do it now. Log into **workstation** as **student** and run the lab setup script.

   ```
   [student@workstation ~]$ lab puppet-satellite setup
   ```

   This creates the **Finance** organization and other Red Hat Satellite entities that are needed to do the lab.

2. Download the **chrony** and **stdlib** Puppet modules to a local directory.

   Launch a web browser and navigate to **http://classroom.example.com/materials/puppet**. Right-click the name of the **chrony** Puppet module and save it to a local file. Do the same for the **stdlib** module.

3. Log into the Satellite web UI and make **Finance** the current organization.

   Log into the Satellite web UI as *admin*. Select **Finance** as the current organization. Click the **Any Context** tab at the left and select **Finance** from the pulldown menu.

4. Create a product called **Finance Product** to contain the Puppet repository.

   Select the **Content** > **Products** tab. Click the **New Product** button.

   Complete the form that appears. Fill in the fields according to the following table:

Field	Value
Name	Finance Product
Label	Finance_Product (autogenerated)

Leave the remaining fields unchanged or blank. Click the **Save** button to confirm your selections and create the product.

5. Create the **`Finance Modules`** Puppet repository.

   Select the **Repositories** tab, then click the **Create Repository** button. Complete the form that appears. Fill in the fields according to the following table:

Field	Value
Name	Finance Modules
Label	Finance_Modules (autogenerated)
Type	puppet

   Leave the remaining fields unchanged or blank. Click the **Save** button to confirm your selections and create the repository.

6. Upload the **`chrony`** and **`stdlib`** Puppet modules into the repository.

   6.1. Select the **Finance Modules** link.

   6.2. In the **Upload Puppet Module** box, click the **Browse...** button.

   6.3. Navigate and select the file that contains the **`chrony`** Puppet module, **`rht-chrony-0.1.0.tar.gz`**.

   6.4. Click the **Upload** button to upload the Puppet module.

   6.5. Repeat these steps for the **`stdlib`** module found in the **`puppetlabs-stdlib-4.9.0.tar.gz`** file.

   6.6. Confirm that the modules specified were uploaded to the server. Click the number to the right of **Puppet Modules** in the **Content Type** section of the page.

7. Create a content view called **`Finance Content`**. It must include the Puppet repository with the **`chrony`** and **`stdlib`** Puppet modules. Publish and promote the content view so it is available to Finance hosts that are assigned to the **`Dev`** software lifecycle environment.

   7.1. Log into the Satellite web interface as the *admin* user. Set the organization context to **`Finance`**. Select **Any Context** > **Any Organization** > **Finance**.

   7.2. Select **Content** > **Content Views**, then click the **Create New View** button. Complete the form that appears with the following values:

Field	Value
Name	Finance Content
Label	Finance_Content (autogenerated)

Field	Value
Description	Content view for Finance servers
Composite View?	Unchecked

Click the **Save** button to confirm.

7.3. Click the **Puppet Modules** tab, then click the **Add New Module** button.

Click the **Select a Version** button that corresponds to the **chrony** module in the **Actions** column. Click the **Select Version** button for the **Use Latest (currently 0.1.0)** version of the **chrony** module.

Repeat the previous steps to use the latest version of the the **stdlib** Puppet module.

7.4. Publish the content view and promote it to the **Dev** environment. Click the **Publish New Version** button, then click **Save** to confirm that you want to publish it.

7.5. When the content view publish completes, click the **Promote** button in the **Actions** column for version 1.0 of the content view. Select the **Dev** environment, then click the **Promote Version** button.

When the promotion completes, both **Library** and **Dev** should be listed in the **Environments** column. This confirms that the publish and promotion succeeded.

8. Confirm the proper locations are assigned to the Puppet environments for the **Finance** organization.

8.1. Make sure **Finance** is the default organization context. Select **Any Context** > **Any Organization** > **Finance**.

8.2. Navigate to the **Puppet Environments** screen by selecting **Configure** > **Environments**.

8.3. Select the environment for the **Finance** content library, **KT_Finance_Library_Finance_Content_#**.

8.4. Click the **Locations** subtab and put **Boston** and **Default Location** in the **Selected items** window.

8.5. Click **Submit** to apply your changes.

8.6. Repeat the previous steps for the **Dev** Puppet environment of the **Finance** organization.

9. Create an activation key in the **Finance** organization, called **Finance Host**, to register Puppet client machines. The **Finance Content** content view should be assigned to hosts that register with it and their software lifecycle environment should be **Dev**. Since the classroom Red Hat Satellite server does not provide software packages, the activation key should not auto attach to any Red Hat software subscriptions. The activation key should subscribe hosts to the custom **Finance Product** product so they have access to the Puppet repository it provides.

9.1. Make sure **Finance** is the default organization context. Select **Any Context** > **Any Organization** > **Finance**.

9.2. Select **Content** > **Activation keys**, then click the **New Activation Key** button. Complete the form that appears with the following values:

Field	Value
Name	Finance Host
Content Host Limit: Unlimited Content Hosts	Leave checked
Description	Activation key that registers Finance development servers
Environment	Select **Dev**
Content View	Select **Finance Content** from the pulldown menu

Click **Save** to confirm.

9.3. Select the **Subscriptions** tab, change **Auto-Attach** to **No**, then click **Save**.

Click the **Add** subtab, then select the **Finance Product** subscription. Click the **Add Selected** button to confirm.

10. Configure the Puppet client on **serverb** for Red Hat Satellite use. Register it to the **Finance** organization with the **Finance Host** activation key.

```
[root@serverb ~]# yum -y install http://satellite.lab.example.com/pub/katello-ca-
consumer-latest.noarch.rpm
... Output omitted ...
[root@serverb ~]# subscription-manager register --org='Finance' \
                    --activationkey='Finance Host'
The system has been registered with ID: 4cbf205c-96ca-4524-b1f3-1b1b37ed7ab4
Installed Product Current Status:
Product Name: Red Hat Enterprise Linux Server
Status:       Not Subscribed

Unable to find available subscriptions for all your installed products.
```

11. Install Puppet and the Katello agent software on the Satellite client host, **serverb**.

```
[root@serverb ~]# yum -y install puppet katello-agent
```

12. Configure Puppet on **serverb** to use the Red Hat Satellite server as the Puppet master. Run **puppet** once in test mode so it sends a certificate to the Satellite server.

If **/etc/puppet/puppet.conf** has a **server=** line, modify it to point to the Satellite server. Otherwise, add a **server=** directive to the configuration file.

```
[root@serverb ~]# echo 'server=satellite.lab.example.com' >> /etc/puppet/puppet.conf
[root@serverb ~]# puppet agent --test
```

13. Use the Satellite web user interface to sign the host certificate for the client system.

13.1. Select **Infrastructure** > **Capsules**.

13.2. Click the **Certificates** button at the right of the **satellite.lab.example.com** capsule host name. A list of host certificates will appear. Click the **Sign** button at the right of the **serverb** host name.

13.3. On **serverb**, use the **puppet** command to check in with the Satellite server to retrieve the client's signed host certificate.

```
[root@serverb ~]# puppet agent --test --noop
```

14. Create a host group called **Finance Host Group** in the **Finance** organization.

14.1. Make sure **Finance** is the default organization context. Select **Any Context** > **Any Organization** > **Finance**.

14.2. Select **Configure** > **Host groups**, then click the **New Host Group** button. Complete the form that appears with the following values:

Field	Value
Name	Finance Host Group
Lifecycle Environment	Dev
Content View	Finance Content
Puppet Environment	KT_Finance_Dev_Finance_Content_#
Content Source	satellite.lab.example.com
Puppet CA	satellite.lab.example.com
Puppet Master	satellite.lab.example.com

14.3. Select the **Puppet Classes** tab, then sElect the **chrony**, **chrony::params**, and **stdlib** classes so they appear in the **Included Classes** column.

14.4. Select the **Locations** tab. Move **Boston** and **Default Location** to the list of **Selected items**.

14.5. Select the **Organizations** tab. Make sure **Finance** is selected.

14.6. Click the **Submit** button at the bottom to confirm selections and create the new host group.

15. Assign the **Finance Host Group** host group profile to **serverb.lab.example.com**. **serverb** should be assigned to the **Finance** organization and the **Boston** location. Apply the **Finance** content view to the host and make it a member of the **Dev** lifecycle environment. It should also be assigned to the **KT_Finance_Dev_Finance_Content_#** Puppet environment. The Satellite server should act as **serverb**'s content source, Puppet certificate authority, and Puppet master.

15.1. Set **Any Organization** as the default organization context. Select **Any Context** > **Finance** > **Any Organization**.

15.2. Select the **Hosts** > **All hosts** tab. Identify **serverb.lab.example.com** in the list of hosts. Note that the **Host group** column is empty. Click the checkbox selecting this host.

15.3. Click the **Select Action** pulldown menu, and choose the **Assign Organization** menu item. Click the **Select Organization** button, choose **Finance**, then click the **Fix Organization on Mismatch** button to confirm the change is effective. Click **Submit** to commit the selections.

15.4. Click the **Select Action** pulldown menu, and choose the **Assign Location** menu item. Click the **Select Location** button, choose **Boston**, click the **Fix Location on Mismatch** button, then click **Submit** to commit the selections.

15.5. Click the **Select Action** pulldown menu, and choose the **Change Environment** menu item. Click the **Select environment** button, choose **KT_Finance_Dev_Finance_Content_#**, then click **Submit** to commit the selection.

15.6. Click the **serverb.lab.example.com** host name link in the list. This will bring up the Puppet details page for that server.

Click the **Edit** button at the top of the screen. Select the following values from the pulldown menus:

Field	Value
Host Group	Finance Host Group
Lifecycle Environment	Dev
Content View	Finance Content
Puppet Environment	KT_Finance_Dev_Finance_Content_#
Content Source	**satellite.lab.example.com**
Puppet CA	**satellite.lab.example.com**
Puppet Master	**satellite.lab.example.com**

Click the **Submit** button to confirm the selections.

16. Define a smart class parameter that populates the **servers** parameter of the **chrony** Puppet class with a list of servers that only includes the **classroom.example.com** and **satellite.lab.example.com** hosts. This configuration should only apply to hosts that are members of the **Finance Host Group** host group.

16.1. Make sure **Finance** is the default organization context. Select **Any Context** > **Any Organization** > **Finance**.

16.2. Select **Configure** > **Puppet classes**, then click **chrony** in the **Class name** column.

16.3. Select the **Smart Class Parameter** subtab. Scroll down, locate, and click the link for the **servers** class parameter at the left. Check the **Override** box to allow for a custom value to be entered.

Keep the default value for the **servers** smart class parameter. Puppet needs to be configured to apply the specified configuration for servers in the **Finance** department only.

16.4. Click the **Add Matcher-Value** button. Complete the form that appears with the following values:

Field	Value
Match	hostgroup=Finance Host Group
Value	["classroom.example.com","satellite.lab.example.com"]

Click the **Submit** button to confirm the changes.

The **`Matcher-Value`** pair that you defined correctly sets the NTP servers only for **`Finance Host Group`** type hosts.

17. Have Puppet apply the change to the host system to confirm the configuration is correct.

```
[root@serverb ~]# puppet agent --test
... Output omitted ...
Info: Applying configuration version '1431439702'
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content:
--- /etc/chrony.conf    2015-05-12 05:43:32.585655052 +0000
+++ /tmp/puppet-file20150512-1226-zzpdwj     2015-05-12 14:08:24.518777983 +0000
@@ -1,4 +1,5 @@
-server 0.rhel.pool.ntp.org iburst
+server classroom.example.com iburst
+server satellite.lab.example.com iburst


 # Ignore stratum in source selection.

Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Filebucketed
/etc/chrony.conf to puppet with sum d90283ded230b6354ca9f4afe8361edd
Notice: /Stage[main]/Chrony/File[/etc/chrony.conf]/content:
content changed '{md5}d90283ded230b6354ca9f4afe8361edd' to
'{md5}051dd44a2dff92e14e2cb19eb90876c8'
Info: /Stage[main]/Chrony/File[/etc/chrony.conf]: Scheduling refresh
of Service[chronyd]
Notice: /Stage[main]/Chrony/Service[chronyd]: Triggered 'refresh' from 1 events
Notice: Finished catalog run in 1.06 seconds
```

18. Persistently configure Puppet on **`serverb`** to run as a service.

```
[root@serverb ~]# systemctl start puppet.service
[root@serverb ~]# systemctl enable puppet.service
ln -s '/usr/lib/systemd/system/puppet.service' '/etc/systemd/multi-
user.target.wants/puppet.service'
```

19. Run the lab grading script from **`workstation`** to check your work. The grading program will access **`serverb`** to evaluate your work.

```
[student@workstation ~]$ lab puppet-satellite grade
```

Summary

In this chapter, you learned:

- Red Hat Satellite 6 can make Puppet modules available to client systems. A product must be created that includes a repository that is defined with a **puppet** type.
- When using Puppet with Red Hat Satellite 6, content views must be defined that publish the Puppet repositories (and additional software repositories) that the Puppet modules may use to install needed software.
- Activation keys can be defined that automate the registration of client systems.
- The *katello-ca-consumer-latest* package, published by the Satellite server, configures the client to use the Satellite server for software. It must be installed before **subscription-manager** is used to register the system.
- Adding **server=*satellite.FQDN*** to **/etc/puppet/puppet.conf** configures Puppet to work with Red Hat Satellite 6.
- The **Certificates** button in **Infrastructure** > **Capsules** is used to sign Puppet agent certificates.
- The **Configure** > **Host groups** menu is used to define standard system definitions on Red Hat Satellite 6.
- Default values for smart class parameters can be changed when a Puppet module is selected by using the **Configure** > **Puppet Classes** menu.